**南京航空航天大学**

**数据结构课程设计报告**

学生姓名：李应飞

学 号：161610338

学 院：计算机科学与技术学院

专 业：计算机科学与技术

班 级：1616103

指导教师：高航

2018年1月10日

**目 录**

1、购物网站信息管理

2、迷宫问题

3、二叉树的应用

4、Huffman编码与解码

5、无线传感器网络

6、排序算法比较

7、Hash表应用

1. 总结
2. 心得体会

一、购物网站信息管理

1、题目简介

设计一个程序，对商铺信息管理，商铺信息包括：商铺编号，商铺名，信誉度，（商品名称1，价格1，销量1），（商品名称2，价格2，销量2），（商品名称3，价格3，销量3）…。

假设商品名称包括(毛巾，牙刷，肥皂，牙膏，洗发水，沐浴露等)，每个商铺具有其中事先确定若干商品及价格，由文件输入，销量初始为0。

[基本要求]

（1）建立一个单向链表存储所有商铺信息（至少30个），以编号为序，编号从1开始递增，从文件中读取数据，并能将数据存储在文件。商铺信息结点的数据结构自行设计。

（2）可以增、删商铺。增加商铺，编号自动加一，插入链表尾部；删除商铺以编号为准，并修改后续结点的编号，保持编号连续性。

（3）可以增、删选定商铺中的商品，修改商品价格。

（4）查询某一种商品名称，建立一个双向循环链表，结点信息是包含该商品的所有商铺编号，商铺名，信誉度，商品名称，价格，销量，按销量高至低排序，并可逐一显示。

（5）购买某一商铺的商品，修改单向链表中商品的信息的销量。

（6）任何的商铺信息变化，实现文件存储。

1. 数据结构

采用线性表的链式结构，商铺有两个指针，一个指向商铺的第一个物品，一个指向下一个商铺。开始写的时候，我没有用文件，所以都是在链表两个指针下操作的。用文件了，增、删商铺，增、删商品。都可以直接在文件里操作。下面用了一个结构体专门用来存商品。

创建物品文件的结构体：

typedef struct Node//存物品的结构体

{

int code;//商铺编号

char name1[10];//商品名称

int price;//价格

int output;//销量

}Node;

物品和商铺的结构体：

typedef struct Thing//物品

{

char name1[10];//商品名称

int price;//价格

int output;//销量

struct Thing \*next;//下一个物品

}Thing,\*LThing;

typedef struct GNode//商铺

{

int code;//商铺编号

char name[10];//商铺名

int degree;//信誉度

LThing first;//指向物品

struct GNode \*down;//下一个商铺

}GNode,\*LGNode;

查询某一种物品的循环链表结构体：

typedef struct OneNode//循环链表结构体

{

int code;//商铺编号

char name[10];//商铺名

int degree;//信誉度

char name1[10];//商品名称

int price;//价格

int output;//销量

struct OneNode \*prior;

struct OneNode \*next;

} Link,\*Linklist;

1. 算法设计思想
2. 创建物品和商铺及相关文件和查询某一种商品名称

通过对链式结构的不断循环，分别创建商铺和物品文件。两个文件分别创建。并且在创建成功后，可以直接查询各个商铺的某一种物品的信息。

1. 增加商铺

可以直接读取文件里最后一个商铺的编号，然后让商铺编号自增。其余由键盘输入。

也可以将商铺文件里的信息读取出来再增加商铺，然后存文件。

1. 删除商铺

通过两个文件的传递，从而实现删除。

也可以读取出来，然后在链表里删除。

注：这里删除商铺，对应的也需要删除商铺里的商品。

1. 增加物品

可以直接通过追加直接存入文件

也可以将商铺文件里的信息读取出来，传递给链表，再增加物品，然后存文件。

1. 删除物品

可以直接通过查询方式通过两个文件的传递删除物品

也可以将商铺文件里的信息读取出来，传递给链表，再删除物品，然后存文件。

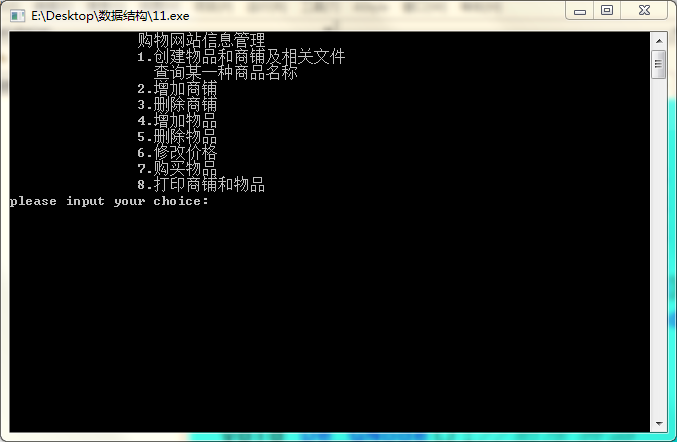
1. 修改价格（7）购买物品

（6）（7）都可以直接通过查询方式通过两个文件的传递修改价格、购买物品

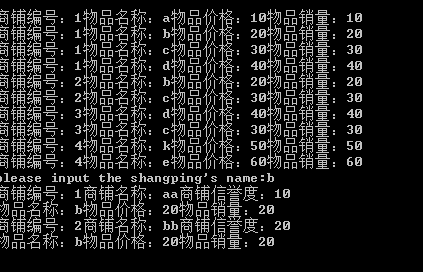
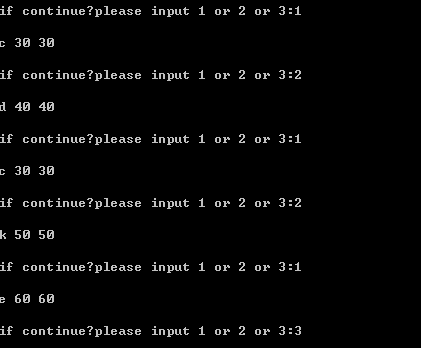
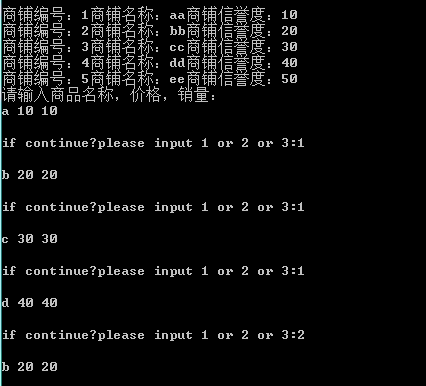
也可以将商铺文件里的信息读取出来，传递给链表，再修改价格、购买物品，然后存文件。

注：因为开始写的时候，我没有用文件，所以其实两种方式都实践过。

1. 测试数据和结果



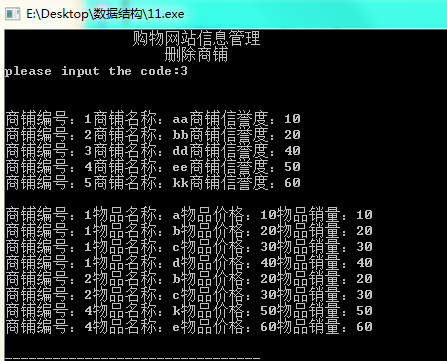
（1）创建物品和商铺及相关文件和查询某一种商品名称



（2）增加商铺



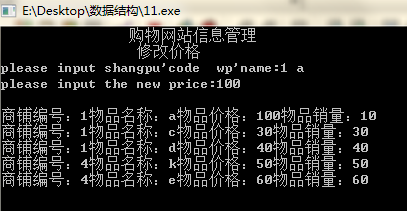
（3）删除商铺



1. 增加物品
2. 删除物品



1. 修改价格



1. 购买物品



1. 算法时间复杂度

算法时间复杂度基本都是0（n）。

改进：查询物品时的循环链表可以改为单链表，简单易操作

1. 源代码

#include<iostream>

using namespace std;

#include<malloc.h>

#include<string.h>

#include<fstream>

#include<windows.h>

typedef struct Node//存物品的结构体

{

int code;//商铺编号

char name1[10];//商品名称

int price;//价格

int output;//销量

}Node;

typedef struct Thing//物品

{

char name1[10];//商品名称

int price;//价格

int output;//销量

struct Thing \*next;//下一个物品

}Thing,\*LThing;

typedef struct GNode//商铺

{

int code;//商铺编号

char name[10];//商铺名

int degree;//信誉度

LThing first;

struct GNode \*down;//下一个商铺

}GNode,\*LGNode;

typedef struct OneNode//循环链表结构体

{

int code;//商铺编号

char name[10];//商铺名

int degree;//信誉度

char name1[10];//商品名称

int price;//价格

int output;//销量

struct OneNode \*prior;

struct OneNode \*next;

} Link,\*Linklist;

void Init\_LinkThing(Linklist &head);//初始化循环链表

void Print\_Onething(Linklist &head);//打印循环链表

void Creat\_Onething(Linklist &head,LGNode head1,char \*c);//创建循环链表查询某一种商品

void Paixu\_Onething(Linklist &head);//排序 按销量

void Init\_Thing(LGNode &head);//创建商铺和物品的空间

void Creat\_Thing(LGNode &head);//创建商铺

void En\_Thing();//增加商铺

void De\_Thing();//删除商铺

void En\_GNode();//增加物品

void De\_GNode();//删除物品

void Creat\_GNode(LGNode &head);////创建物品

void Print\_Thing();//打印商铺和物品

void Eidt\_Price();//修改价格

void Buy\_GNode();//买物品 销量++

void Init\_LinkThing(Linklist &head)//初始化循环链表

{

head=(Linklist)malloc(sizeof(Link));

if(!head)exit(0);

head->next=NULL;

head->prior=NULL;

}

void Print\_Onething(Linklist &head)//打印循环链表

{

Linklist s;

s=head;

int k=1;

while(s&&k)

{

cout<<"商铺编号："<<s->code<<"商铺名称："<<s->name<<"商铺信誉度："<<s->degree<<endl;

cout<<"物品名称："<<s->name1<<"物品价格："<<s->price<<"物品销量："<<s->output<<endl;

s=s->next;

if(s==head)

k=0;

}

}

void Creat\_Onething(Linklist &head,LGNode head1,char \*c)//创建循环链表查询某一种商品

{

LGNode q,q1;

LThing p,p1;

Linklist s,s1;

if(!head)exit(0);

if(!head1)exit(0);

q=head1->down;

int key=0;

while(q)

{

p=q->first;

while(p)

{

if(!strcmp(p->name1,c))

{

if(key==0)

{

head->code=q->code;

head->degree=q->degree;

strcpy(head->name,q->name);

strcpy(head->name1,p->name1);

head->price=p->price;

head->output=p->price;

head->next=head->prior=head;

key++;

s=head;

}

else

{

s1=s;

s=(Linklist)malloc(sizeof(Link));

s->code=q->code;

s->degree=q->degree;

strcpy(s->name,q->name);

strcpy(s->name1,p->name1);

s->price=p->price;

s->output=p->output;

s1->next=s;

s->next=head;

s->prior=s1;

head->prior=s;

}

break;

}

else

{

p=p->next;

}

}

q=q->down;

}

}

void Paixu\_Onething(Linklist &head)//排序 按销量

{//简单排序

Linklist s,s1,s2;

if(!head)exit(0);

if(head->next)

s=head->next;

s2=head;

while(s)

{

if(s2->output<=s->output)

{

s1=head;

while(s1&&(s1!=s))

{

if(s->output<s1->output)

{

s1->next=s;

s->prior=s1;

s->next=s1->next;

s1->next->prior=s;

}

else

{

s1=s1->next;

}

}

}

s2=s2->next;

s=s->next;

}

}

void Init\_Thing(LGNode &head)//创建商铺和物品的空间

{

head=(LGNode)malloc(sizeof(GNode));

if(!head)exit(0);

head->down=NULL;

}

void Creat\_Thing(LGNode &head)//创建商铺

{

fstream file("shangpu.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

cout<<"请输入商铺（商铺编号，商铺名，信誉度）："<<endl;

LGNode p,q;

GNode f;

p=head;

int key=1;

while(key)

{

q=(LGNode)malloc(sizeof(GNode));

cin>>q->code;//输入商铺编号

cin>>q->name;//输入商铺名

cin>>q->degree;//输入信誉度

q->first=NULL;//物品指针置空

q->down=NULL;

f.code=q->code;

f.degree=q->degree;

strcpy(f.name,q->name);

f.first=q->first;

f.down=q->down;

file.write((char \*)&f,sizeof(f));

p->down=q;//head为空

p=q;

int data;//判断是否继续输入

cout<<endl;

cout<<"if continue , please input 1 or 0.: ";

cin>>data;

cout<<endl;

if(data==0)

{

break;

}

}

file.close();

cout<<endl;

file.open("shangpu.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<"商铺编号："<<f.code<<"商铺名称："<<f.name<<"商铺信誉度："<<f.degree<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

}

void En\_Thing( )//增加商铺

{

fstream file;

GNode f;

cout<<endl;

file.open("shangpu.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<"商铺编号："<<f.code<<"商铺名称："<<f.name<<"商铺信誉度："<<f.degree<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

file.open("shangpu.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

int k;

while(!file.eof())

{

k=f.code;

file.read((char \*)&f,sizeof(f));

}

file.close();

file.open("shangpu.dat",ios::app|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

GNode ff;

cout<<"please input 商铺名 信誉度:";

k++;

ff.code=k;

cin>>ff.name;

cin>>ff.degree;

ff.first=NULL;

ff.down=NULL;

file.write((char \*)&ff,sizeof(ff));

file.close();

cout<<endl;

file.open("shangpu.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<"商铺编号："<<f.code<<"商铺名称："<<f.name<<"商铺信誉度："<<f.degree<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

}

void De\_Thing\_GN(int code)//删除商铺 对应的删除物品

{

fstream file,file1;

Node f;

file.open("wuping.dat",ios::in|ios::binary);

file1.open("wuping1.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

if(file1.fail())

{

cout<<"file1 open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));//删除商铺

while(!file.eof())

{

if(f.code!=code)

file1.write((char \*)&f,sizeof(f));

file.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

file.open("wuping.dat",ios::out|ios::binary);//文件传递

file1.open("wuping1.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

if(file1.fail())

{

cout<<"file1 open is error!"<<endl;

exit(0);

}

file1.read((char \*)&f,sizeof(f));

while(!file1.eof())

{

file.write((char \*)&f,sizeof(f));

file1.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

cout<<endl;

file.open("wuping.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<"商铺编号："<<f.code<<"物品名称："<<f.name1<<"物品价格："<<f.price<<"物品销量："<<f.output<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

}

void De\_Thing()//删除商铺

{//两个文件

int code;

cout<<"please input the code:";

cin>>code;

fstream file,file1;

GNode f;

cout<<endl;

file.open("shangpu.dat",ios::in|ios::binary);

file1.open("shangpu1.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

if(file1.fail())

{

cout<<"file1 open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));//删除商铺

while(!file.eof())

{

if(f.code!=code)

{

if(f.code>code)//商铺编号--

f.code--;

file1.write((char \*)&f,sizeof(f));

}

file.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

file.open("shangpu.dat",ios::out|ios::binary);//文件传递

file1.open("shangpu1.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

if(file1.fail())

{

cout<<"file1 open is error!"<<endl;

exit(0);

}

file1.read((char \*)&f,sizeof(f));

while(!file1.eof())

{

file.write((char \*)&f,sizeof(f));

file1.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

cout<<endl;//打开文件检测

file.open("shangpu.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<"商铺编号："<<f.code<<"商铺名称："<<f.name<<"商铺信誉度："<<f.degree<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

De\_Thing\_GN(code);

}

void En\_GNode()//增加物品

{

fstream file("wuping.dat",ios::app|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

Node f;

cout<<"please input shangpu's code:";

cin>>f.code;

cout<<"请输入商品名称，价格，销量："<<endl;

cin>>f.name1;

cin>>f.output;

cin>>f.price;

file.write((char \*)&f,sizeof(f));

file.close();

cout<<endl;

file.open("wuping.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<"商铺编号："<<f.code<<"物品名称："<<f.name1<<"物品价格："<<f.price<<"物品销量："<<f.output<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

}

void De\_GNode()

///void De\_GNode(LGNode &head,char \*c)//删除物品

{

fstream file,file1;

Node f;

int code;

char name[10];

cout<<"please input shangpu'code wp'name:";

cin>>code;

cin>>name;

file.open("wuping.dat",ios::in|ios::binary);

file1.open("wuping1.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

if(file1.fail())

{

cout<<"file1 open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));//删除物品

while(!file.eof())

{

if(f.code!=code&&strcmp(f.name1,name))

file1.write((char \*)&f,sizeof(f));

file.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

file.open("wuping.dat",ios::out|ios::binary);//文件传递

file1.open("wuping1.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

if(file1.fail())

{

cout<<"file1 open is error!"<<endl;

exit(0);

}

file1.read((char \*)&f,sizeof(f));

while(!file1.eof())

{

file.write((char \*)&f,sizeof(f));

file1.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

cout<<endl;

file.open("wuping.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<"商铺编号："<<f.code<<"物品名称："<<f.name1<<"物品价格："<<f.price<<"物品销量："<<f.output<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

/\* LGNode q;

if(!head->down)exit(0);

q=head->down;

LThing p,p1;

while(q)

{

p1=p=q->first;

if(!strcmp(p->name1,c))

{

q->first=p->next;

free(p);

}

else

{

p=p->next;

while(p)

{

if(!strcmp(p->name1,c))

{

p1->next=p->next;

free(p);

break;

}

else

{

p=p->next;

p1=p1->next;

}

}

}

q=q->down;

}\*/

}

void Creat\_GNode(LGNode &head)////创建物品

{

fstream file("wuping.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

Node f;

LGNode q;

if(!head->down)exit(0);

q=head->down;

//cout<<q->code<<endl;

cout<<"请输入商品名称，价格，销量："<<endl;

int key1=1;

LThing T,p1;//开辟空间存物品信息

char n[12];

int pr,out;

while(key1)

{

cin>>n;

cin>>pr>>out;

f.code=q->code;

T=(Thing \*)malloc(sizeof(Thing));

strcpy(T->name1,n);//商品名称

strcpy(f.name1,n);

T->price=pr;//价格

f.price=pr;

T->output=out;//销量

f.output=out;

T->next=NULL;

//尾插法

file.write((char \*)&f,sizeof(f));

if(q->first==NULL)

q->first=T;//first不空

else

{

p1=q->first;

while(p1->next)

p1=p1->next;

p1->next=T;

}

//cout<<"2"<<endl;

int data;

cout<<endl;

cout<<"if continue?please input 1 or 2 or 3:";

cin>>data;

cout<<endl;

switch(data)

{

case 1://继续添加物品

{

break;

}

case 2://添加下一个商铺物品信息

{

q=q->down;

if(!q)

key1=0;

break;

}

case 3://结束

{

key1=0;

break;

}

}

}

file.close();

cout<<endl;

file.open("wuping.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<"商铺编号："<<f.code<<"物品名称："<<f.name1<<"物品价格："<<f.price<<"物品销量："<<f.output<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

}

void Print\_Thing()

//void Print\_Thing(LGNode head)//打印商铺和物品

{

fstream file;

GNode s;

file.open("shangpu.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&s,sizeof(s));

while(!file.eof())

{

cout<<"商铺编号："<<s.code<<"商铺名称："<<s.name<<"商铺信誉度："<<s.degree<<endl;

file.read((char \*)&s,sizeof(s));

}

file.close();

Node f;

file.open("wuping.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<"商铺编号："<<f.code<<"物品名称："<<f.name1<<"物品价格："<<f.price<<"物品销量："<<f.output<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

/\* LGNode p;

p=head->down;

if(p==NULL)exit(0);

while(p)

{

cout<<"商铺编号："<<p->code<<"商铺名称："<<p->name<<"商铺信誉度："<<p->degree<<endl;

LThing T;

T=p->first;

while(T)

{

cout<<"物品名称："<<T->name1<<"物品价格："<<T->price<<"物品销量："<<T->output<<endl;

T=T->next;

}

p=p->down;

}\*/

}

void Eidt\_Price()

//void Eidt\_Price(LGNode &head,char \*a,int pr)//修改价格

{

fstream file,file1;

Node f;

int code;

char name[10];

cout<<"please input shangpu'code wp'name:";

cin>>code;

cin>>name;

file.open("wuping.dat",ios::in|ios::binary);

file1.open("wuping1.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

if(file1.fail())

{

cout<<"file1 open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

if(f.code==code&&!strcmp(f.name1,name))

{

int pr;

cout<<"please input the new price:";

cin>>pr;

f.price=pr;

}

file1.write((char \*)&f,sizeof(f));

file.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

file.open("wuping.dat",ios::out|ios::binary);//文件传递

file1.open("wuping1.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

if(file1.fail())

{

cout<<"file1 open is error!"<<endl;

exit(0);

}

file1.read((char \*)&f,sizeof(f));

while(!file1.eof())

{

file.write((char \*)&f,sizeof(f));

file1.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

cout<<endl;

file.open("wuping.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<"商铺编号："<<f.code<<"物品名称："<<f.name1<<"物品价格："<<f.price<<"物品销量："<<f.output<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

/\* LGNode q;

if(!head->down)exit(0);

q=head->down;

LThing p;

while(q)

{

p=q->first;

if(!strcmp(p->name1,a))

{

p->price=pr;

}

else

{

p=p->next;

while(p)

{

if(!strcmp(p->name1,a))

{

p->price=pr;

break;

}

else

{

p=p->next;

}

}

}

q=q->down;

}\*/

}

void Buy\_GNode()

//void Buy\_GNode(LGNode &head,int c,char \*a)//买物品 销量++

{

fstream file,file1;

Node f;

int code;

char name[10];

cout<<"please input shangpu'code wp'name:";

cin>>code;

cin>>name;

file.open("wuping.dat",ios::in|ios::binary);

file1.open("wuping1.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

if(file1.fail())

{

cout<<"file1 open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));//删除物品

while(!file.eof())

{

if(f.code==code&&!strcmp(f.name1,name))

f.output++;

file1.write((char \*)&f,sizeof(f));

file.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

file.open("wuping.dat",ios::out|ios::binary);//文件传递

file1.open("wuping1.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

if(file1.fail())

{

cout<<"file1 open is error!"<<endl;

exit(0);

}

file1.read((char \*)&f,sizeof(f));

while(!file1.eof())

{

file.write((char \*)&f,sizeof(f));

file1.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

cout<<endl;

file.open("wuping.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<"商铺编号："<<f.code<<"物品名称："<<f.name1<<"物品价格："<<f.price<<"物品销量："<<f.output<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

/\* LGNode q;

if(!head->down)exit(0);

q=head->down;

LThing p;

while(q)

{

if(q->code==c)

{

p=q->first;

while(p)

{

if(!strcmp(p->name1,a))

{

p->output++;

break;

}

else

p=p->next;

}

break;

}

else

q=q->down;

}\*/

}

int main()

{

LGNode head;

Init\_Thing(head);//创建商铺和物品的空间

cout<<" 购物网站信息管理"<<endl;

cout<<" 1.创建物品和商铺及相关文件 "<<endl;

cout<<" 查询某一种商品名称"<<endl;

cout<<" 2.增加商铺 "<<endl;

cout<<" 3.删除商铺 "<<endl;

cout<<" 4.增加物品"<<endl;

cout<<" 5.删除物品"<<endl;

cout<<" 6.修改价格"<<endl;

cout<<" 7.购买物品"<<endl;

cout<<" 8.打印商铺和物品 "<<endl;

int choice;

cout<<"please input your choice:"<<endl;

cin>>choice;

switch(choice)

{

case 1:

system("cls");

cout<<" 购物网站信息管理"<<endl;

cout<<" 创建物品和商铺 "<<endl;

cout<<" 查询某一种商品名称"<<endl;

Creat\_Thing(head);

Creat\_GNode(head);

Linklist T;

cout<<"please input the shangping's name:";

char name2[10];

cin>>name2;

Init\_LinkThing(T);

Creat\_Onething(T,head,name2);

Print\_Onething(T);

break;

case 2:

system("cls");

cout<<" 购物网站信息管理"<<endl;

cout<<" 增加商铺 "<<endl;

En\_Thing();

break;

case 3:

system("cls");

cout<<" 购物网站信息管理"<<endl;

cout<<" 删除商铺 "<<endl;

De\_Thing();

break;

case 4:

system("cls");

cout<<" 购物网站信息管理"<<endl;

cout<<" 增加物品"<<endl;

En\_GNode();

break;

case 5:

system("cls");

cout<<" 购物网站信息管理"<<endl;

cout<<" 删除物品"<<endl;

De\_GNode();

break;

case 6:

system("cls");

cout<<" 购物网站信息管理"<<endl;

cout<<" 修改价格"<<endl;

Eidt\_Price();

break;

case 7:

system("cls");

cout<<" 购物网站信息管理"<<endl;

cout<<" 购买物品"<<endl;

Buy\_GNode();

break;

case 8:

Print\_Thing();

break;

default:

cout<<"input data is error!";

}

}

二、迷宫问题

1、题目简介

[问题描述]

利用栈操作实现迷宫问题求解。(选做)（深度搜索，广度搜索）；

[基本要求]

（1）从文件中读取数据，生成模拟迷宫地图，不少于10行10列。

（2）给出任意入口和出口，显示输出迷宫路线。

2、数据结构

判断四个方向，实现不断入栈出栈；

深度搜索（递归），广度搜索（队列）

#define STACK\_INIT\_SIZE 100

#define STACKINCREAT 10

创建地图结构体：

typedef struct

{

int i;//坐标

int j;//坐标

int weight;//标记

}map;

用栈实现迷宫结构体：

typedef struct

{

int i;//坐标

int j;//坐标

int dir;//方向

}Pos;

typedef struct

{

Pos \*base;

Pos \*top;

int stacksize;

}SqStack;

队列：

typedef struct QNode

{

int x;

int y;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct

{

QueuePtr front;//队头指针

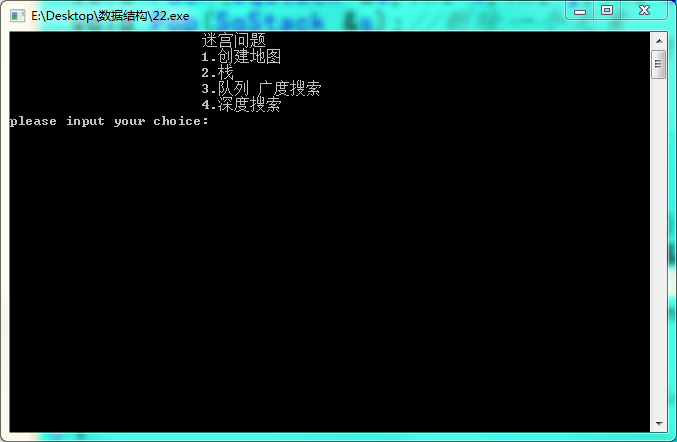
QueuePtr rear;//队尾指针

}LinkQueue;

1. 算法设计思想

判断四个方向，实现不断入栈出栈，并做好标记；

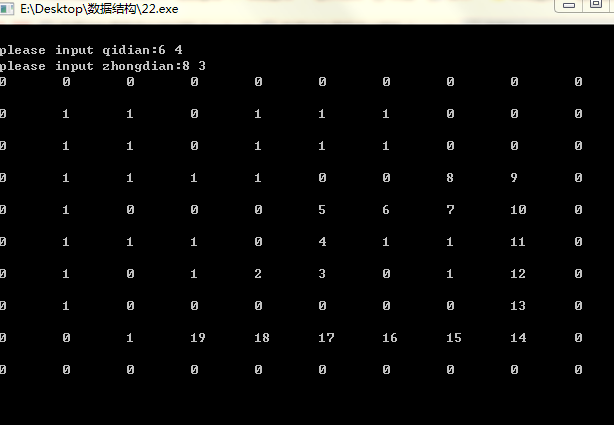
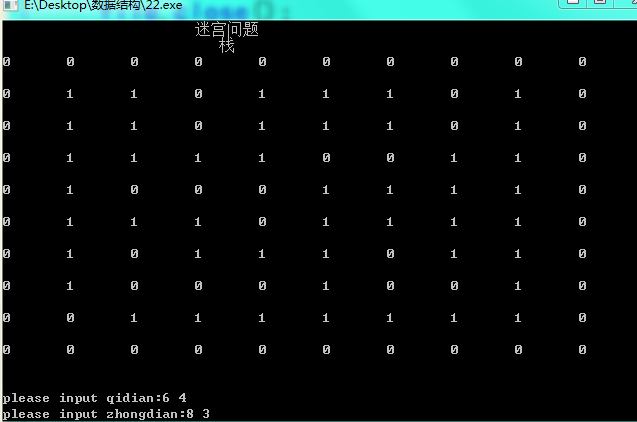
深度搜索（递归），广度搜索（队列），并做好标记；（与图的遍历相似）

1. 测试数据和结果

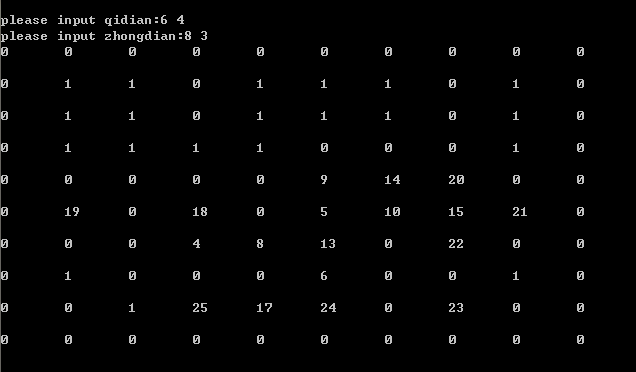
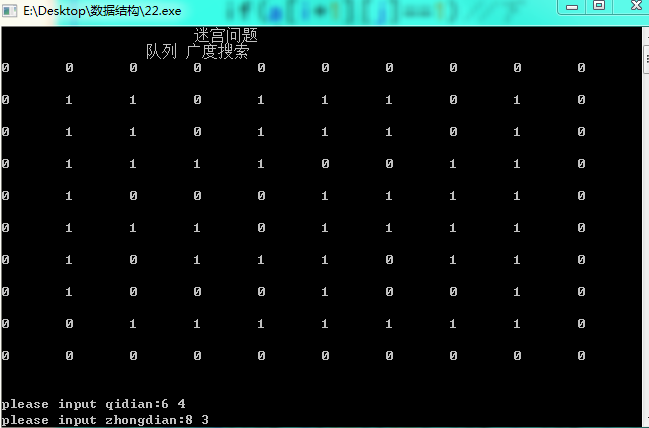
主要要求的是路径，我的后面几种方法只显示最终过程，其实可以求路径，只需在程序中输出走过的路。

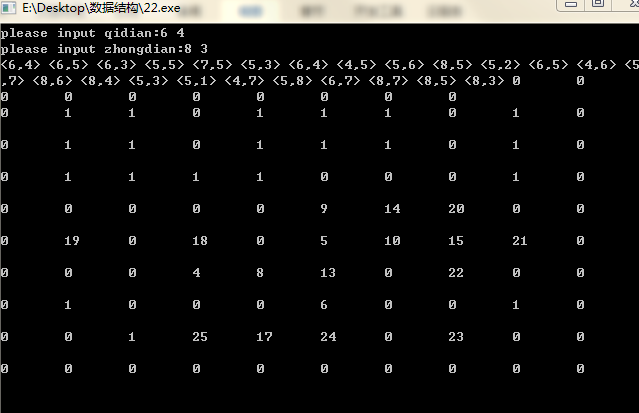
1. 栈：起点 终点键盘输入

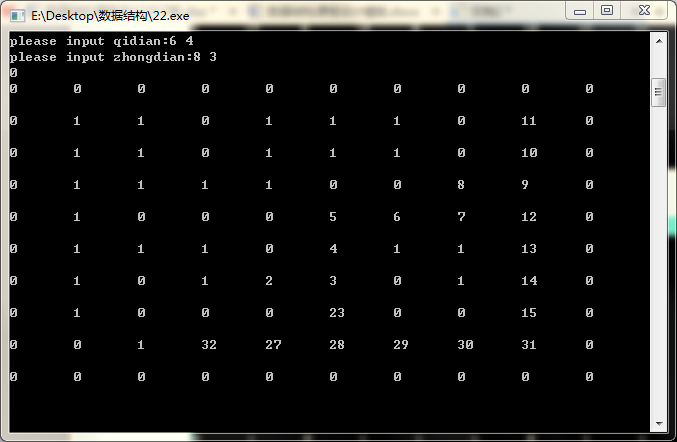
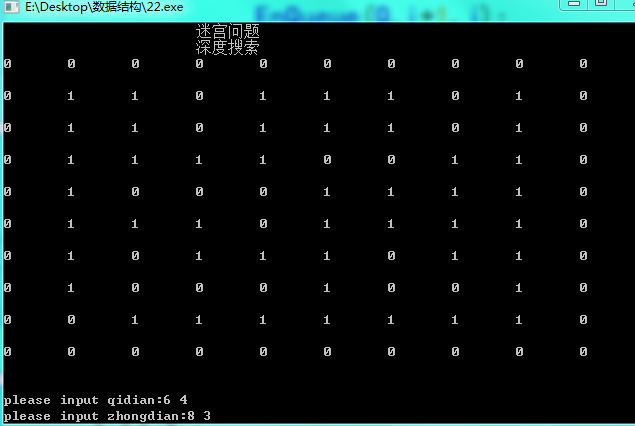
图中2代表起点位置

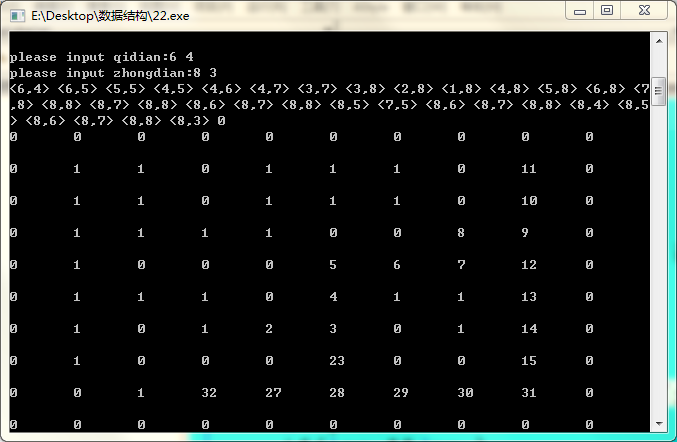


1. 队列 广度搜索：最大值即终点



改进：（开始我想的是只要找到终点及一些相关路径就可以了，在验收时高老师说要每一步的路径，这里我改进了，显示出坐标变化）

1. 深度搜索

改进：（开始我想的是只要找到终点及一些相关路径就可以了，在验收时高老师说要每一步的路径，这里我改进了，显示出坐标变化）

1. 算法时间复杂度

时间复杂度基本是0（n）；

改进：图像界面。和不改变迷宫里的权值。

6、源代码

#include<iostream>

using namespace std;

#include<malloc.h>

#include<fstream>

#define STACK\_INIT\_SIZE 100

#define STACKINCREAT 10

typedef struct

{

int i;

int j;

int weight;

}map;

typedef struct

{

int i;

int j;

int dir;

}Pos;

typedef struct

{

Pos \*base;

Pos \*top;

int stacksize;

}SqStack;

typedef struct QNode

{

int x;

int y;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct

{

QueuePtr front;//队头指针

QueuePtr rear;//队尾指针

}LinkQueue;

void InitStack(SqStack &s);//构造空栈和赋值

void Print\_Stack(SqStack s);//打印栈

void Push(SqStack &s,int x,int y,int dir);//添加一个元素

void Pop(SqStack &s);//删除一个元素

int GetTop(SqStack s);//得到栈顶的元素

void print(int a[10][10]);

void Solution(SqStack &L);//栈

void InitQueue(LinkQueue &Q);

void QueueTraverse(LinkQueue Q);

int QueueEmpty(LinkQueue Q);

void EnQueue(LinkQueue &Q,int x,int y);

void DeQueue(LinkQueue &Q,int &x,int &y);

void QSolution(LinkQueue &Q)；//队列

int Find(int a[10][10],int i,int j,int x,int y,int &k,int &flag);//深度搜索

void Creat\_map();//创建地图

void Solution3();//深度搜索

void InitStack(SqStack &s)//构造空栈和赋值

{

s.base=(Pos \*)malloc(STACK\_INIT\_SIZE\*sizeof(Pos));

if(!s.base)exit(0);

s.top=s.base;

s.stacksize=STACK\_INIT\_SIZE;

}

void Print\_Stack(SqStack s)//打印栈

{

cout<<"the SqStack element:";

cout<<s.top-s.base<<endl;

SqStack s1;

s1.base=s.base;

for(int i=0; i<s.top-s.base; i++)//元素的长度length=s.top-s.base

{

cout<<"i:"<<s1.base->i<<" j:"<<s1.base->j<<" "<<" dir:"<<s1.base->dir<<endl;

s1.base++;

}

cout<<endl;

}

void Push(SqStack &s,int x,int y,int dir)//添加一个元素

{

if(s.top-s.base>=s.stacksize)

{

s.base=(Pos \*)realloc(s.base,(s.stacksize+STACKINCREAT)\*sizeof(Pos));

if(!s.base)exit(0);

s.top=s.base+s.stacksize;

s.stacksize+=STACKINCREAT;

}

s.top->i=x;

s.top->j=y;

s.top->dir=dir;

s.top++;

}

void Pop(SqStack &s)//删除一个元素

{

if(s.base==s.top)exit(0);

--s.top;

}

int GetTop(SqStack s)//得到栈顶的元素

{

if(s.base==s.top)exit(0);

s.top--;

return s.top->dir;

}

void print(int a[10][10])

{

for(int i=0;i<10;i++)

for(int j=0;j<10;j++)

{

cout<<a[i][j]<<" ";

if(j==9)

cout<<endl;

}

cout<<endl;

}

void Solution(SqStack &L)//栈

{

fstream file;

int a[10][10];

map p;

file.open("map.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

file.read((char \*)&p,sizeof(p));

while(!file.eof())

{

a[p.i][p.j]=p.weight;

file.read((char \*)&p,sizeof(p));

}

file.close();

print(a);

int k=2;

int i,j;

cout<<"please input qidian:";

cin>>i>>j;

a[i][j]=k;

int x,y;

cout<<"please input zhongdian:";

cin>>x>>y;

Push(L,i,j,99);

// Print\_Stack(L);

while((i!=x||j!=y))

{

if(a[i-1][j]==1)

{

Push(L,i-1,j,11);

i=i-1;

a[i][j]=k+1;

k++;

}

else if(a[i][j+1]==1)

{

Push(L,i,j+1,22);

j=j+1;

a[i][j]=k+1;

k++;

}

else if(a[i+1][j]==1)

{

Push(L,i+1,j,33);

i=i+1;

a[i][j]=k+1;

k++;

}

else if(a[i][j-1]==1)

{

Push(L,i,j-1,44);

j=j-1;

a[i][j]=k+1;

k++;

}

else

{

a[i][j]=0;

if(GetTop(L)==11)

i=i+1;

if(GetTop(L)==22)

j=j-1;

if(GetTop(L)==33)

i=i-1;

if(GetTop(L)==44)

j=j+1;

Pop(L);

k--;

// print(a);

}

}

print(a);

}

void InitQueue(LinkQueue &Q)

{

Q.front=Q.rear=(QNode \*)malloc(sizeof(QNode));

if(!Q.front)exit(0);

Q.front->next=NULL;

}

void QueueTraverse(LinkQueue Q)

{

QueuePtr L;

if(Q.front->next!=NULL)

{

L=Q.front->next;

cout<<"output all element:";

while(L)

{

cout<<L->x<<" "<<L->y<<" ";

L=L->next;

}

}

}

int QueueEmpty(LinkQueue Q)

{

if(Q.front->next==NULL)

return 1;

else

return 0;

}

void EnQueue(LinkQueue &Q,int x,int y)

{

QueuePtr s;

s=(QNode \*)malloc(sizeof(QNode));

if(!s)exit(0);

s->x=x;

s->y=y;

Q.rear->next=s;

Q.rear=Q.rear->next;

Q.rear->next=NULL;

}

void DeQueue(LinkQueue &Q,int &x,int &y)

{

if(Q.front==Q.rear)exit(0);

QueuePtr s;

s=Q.front->next;

x=s->x;

y=s->y;

Q.front->next=s->next;

if(Q.rear==s)

Q.rear=Q.front;

free(s);

}

void QSolution(LinkQueue &Q)//队列

{

fstream file;

int a[10][10];

map p;

file.open("map.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

file.read((char \*)&p,sizeof(p));

while(!file.eof())

{

a[p.i][p.j]=p.weight;

file.read((char \*)&p,sizeof(p));

}

file.close();

print(a);

int k=2;

int x,y,x1,y1;

cout<<"please input qidian:";

cin>>x>>y;

cout<<"please input zhongdian:";

cin>>x1>>y1;

a[x][y]=k;

int i,j;

EnQueue(Q,x,y);

while(!QueueEmpty(Q))

{

DeQueue(Q,i,j);

a[i][j]=k++;

cout<<"<"<<i<<","<<j<<">"<<" ";

//print(a);

if(i==x1&&j==y1)

break;

if(a[i-1][j]==1)//上

{

EnQueue(Q,i-1,j);

a[i-1][j]=0;

}

if(a[i][j+1])//右

{

EnQueue(Q,i,j+1);

a[i][j+1]=0;

}

if(a[i+1][j]==1)//下

{

EnQueue(Q,i+1,j);

a[i+1][j]=0;

}

if(a[i][j-1]==1)//左

{

EnQueue(Q,i,j-1);

a[i][j-1]=0;

}

}

print(a);

}

int Find(int a[10][10],int i,int j,int x,int y,int &k,int &flag)

{

if(flag)return 0;

a[i][j]=k;

// print(a);

cout<<"<"<<i<<","<<j<<">"<<" ";

k++;

if(i==x&&j==y)

{

flag=1;

return 1;

}

else

{

if(a[i-1][j]==1)//上

{

Find(a,i-1,j,x,y,k,flag);

}

if(a[i][j+1])//右

{

Find(a,i,j+1,x,y,k,flag);

}

if(a[i+1][j]==1)//下

{

Find(a,i+1,j,x,y,k,flag);

}

if(a[i][j-1]==1)//左

{

Find(a,i,j-1,x,y,k,flag);

}

}

}

void Solution3()//深度搜索

{

fstream file;

int a[10][10];

map p;

file.open("map.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

file.read((char \*)&p,sizeof(p));

while(!file.eof())

{

a[p.i][p.j]=p.weight;

file.read((char \*)&p,sizeof(p));

}

file.close();

print(a);

int k=2,flag=0;

int x,y,x1,y1;

cout<<"please input qidian:";

cin>>x>>y;

cout<<"please input zhongdian:";

cin>>x1>>y1;

cout<<Find(a,x,y,x1,y1,k,flag)<<endl;

print(a);

}

void Creat\_map()//创建地图

{//其实可以实现手动输入

int a[10][10];

int k=2;

for(int i=0;i<10;i++)

for(int j=0;j<10;j++)

{

if(i==0||i==9||j==0||j==9)

a[i][j]=0;

else if(j==1&&i==8)

a[i][j]=0;

else if(j==2&&(i==4||i==6||i==7))

a[i][j]=0;

else if(j==3&&(i==1||i==2||i==4||i==7))

a[i][j]=0;

else if(j==4&&(i==4||i==5||i==7))

a[i][j]=0;

else if(j==5&&i==3)

a[i][j]=0;

else if(j==6&&(i==3||i==6||i==7))

a[i][j]=0;

else if(j==7&&(i==1||i==2||i==7))

a[i][j]=0;

else

a[i][j]=1;

}

fstream file;

map p;

file.open("map.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

for(int i=0;i<10;i++)

for(int j=0;j<10;j++)

{

p.i=i;

p.j=j;

p.weight=a[i][j];

file.write((char \*)&p,sizeof(p));

}

file.close();

file.open("map.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

file.read((char \*)&p,sizeof(p));

while(!file.eof())

{

cout<<"i:"<<p.i<<" j:"<<p.j<<" weight:"<<p.weight<<endl;

file.read((char \*)&p,sizeof(p));

}

}

int main()

{

cout<<" 迷宫问题"<<endl;

cout<<" 1.创建地图"<<endl;

cout<<" 2.栈"<<endl;

cout<<" 3.队列 广度搜索"<<endl;

cout<<" 4.深度搜索"<<endl;

int choice;

cout<<"please input your choice:";

cin>>choice;

switch(choice)

{

case 1:

system("cls");

cout<<" 迷宫问题"<<endl;

cout<<" 创建地图"<<endl;

Creat\_map();

break;

case 2:

system("cls");

cout<<" 迷宫问题"<<endl;

cout<<" 栈"<<endl;

SqStack L;

InitStack(L);

Solution(L);

//Print\_Stack(L);

break;

case 3:

system("cls");

cout<<" 迷宫问题"<<endl;

cout<<" 队列 广度搜索"<<endl;

LinkQueue Q;

InitQueue(Q);

QSolution(Q);

//QueueTraverse(Q);

break;

case 4:

system("cls");

cout<<" 迷宫问题"<<endl;

cout<<" 深度搜索"<<endl;

Solution3();

break;

default:

cout<<"input data is error!";

}

return 0;

}

三、二叉树的应用

1、题目简介

[问题描述]

编程实现二叉树的建立，先序、中序、后序（递归和非递归方法）、层序遍历。求二叉树的高度、宽度，结点数。判断是否为二叉排序树。

[基本要求]

（1） 从文件中读入建树信息，树的节点数目不小于20个，树的高度不小于4。

（2） 采用二叉链表结构。

（3） 至少2组输入数据，分别是二叉排序树和不是二叉排序树，

1. 数据结构

#define max 100

#define add 10

typedef struct BiTNode//二叉链表结构体

{

int data;

struct BiTNode \*lchild,\*rchild;//左右孩子指针

}BiTNode,\*BiTree;

typedef struct QNode//队列

{

BiTNode \*data;//（int data存数据BITNode\* data存树的指针）

struct QNode \*next;//下一个数据的指针

}QNode,\*QueuePtr;

typedef struct

{

QueuePtr front;//队列头

QueuePtr rear;//队列尾

}LinkQueue;

typedef struct//栈的结构体

{

BiTNode \* \*base;//栈底指针

BiTNode \* \*top;//栈顶指针

int stacksize;//栈的大小

}Sqstack;

1. 算法设计思想
2. 先序、中序、后序（递归和非递归方法）

1.先序遍历非递归

用栈实现

思想：根入栈，出栈 右子树 左子树入栈

即保证出栈的顺序是按照根 左子树 右子树

1. 中序遍历非递归

用栈实现

思想：不断的将根 左子树入栈，出栈时，右子树入栈

即保证出栈的顺序是按照左子树 根 右子树。

1. 后序遍历非递归

用栈实现

思想：控制出栈的条件是左子树的左右孩子为空，或者是已经访问过

1. 层序遍历。

用队列实现

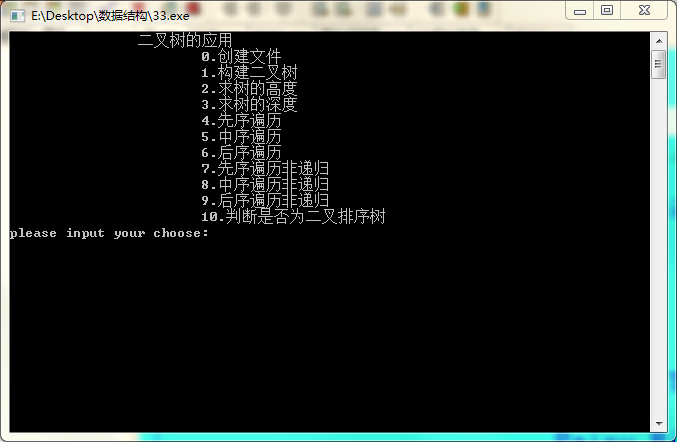
思想：控制入队列是根，左孩子，右孩子。出队列的条件是根，左孩子，右孩子。

1. 求二叉树的高度、宽度，结点数。

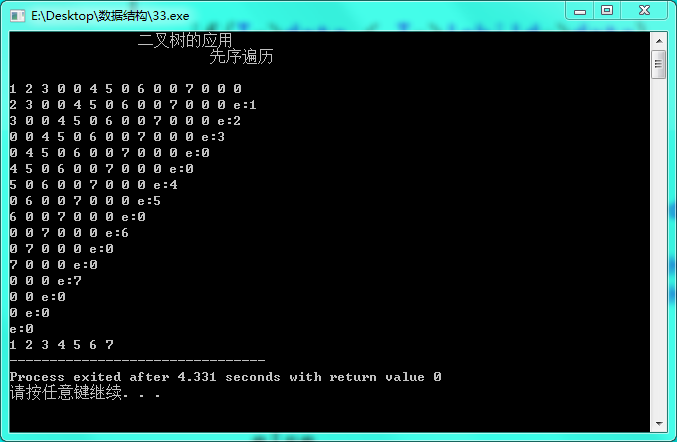
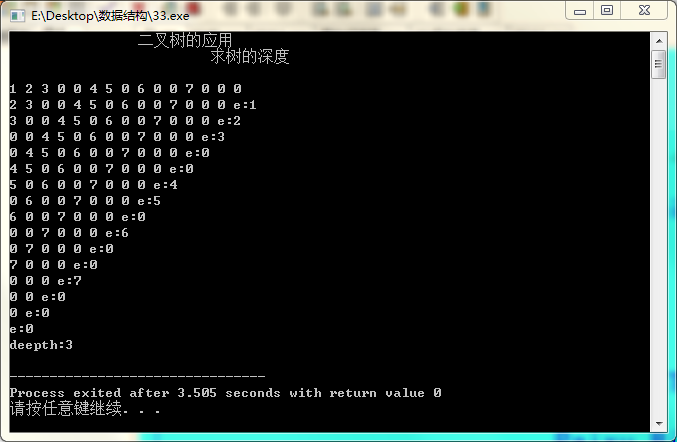
递归实现

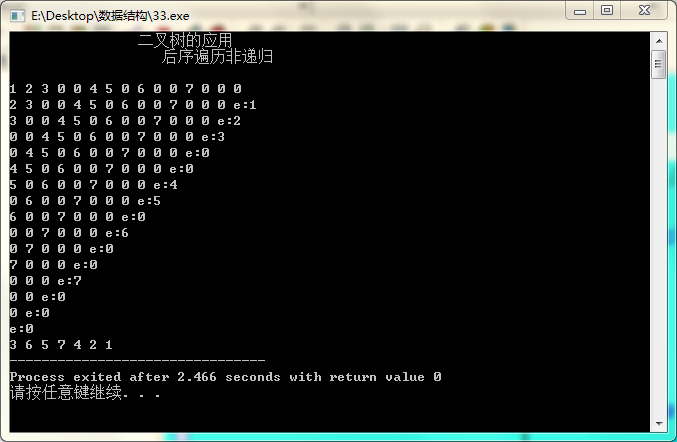
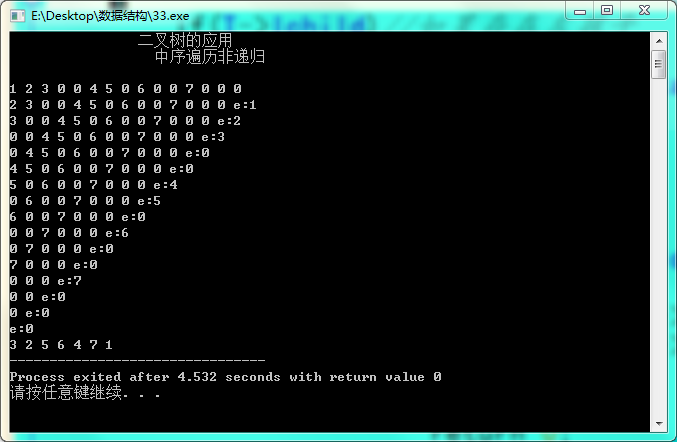
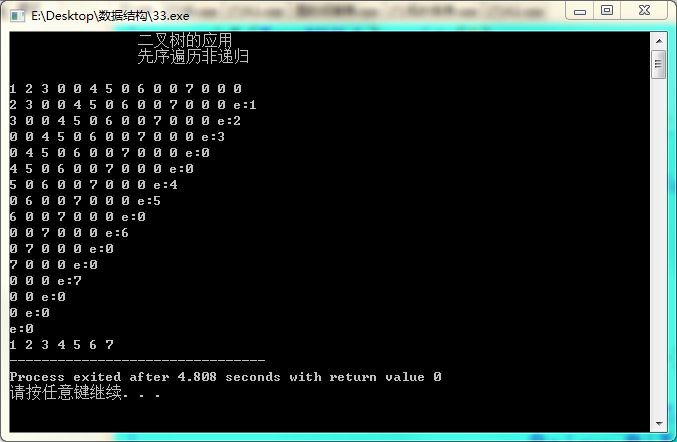
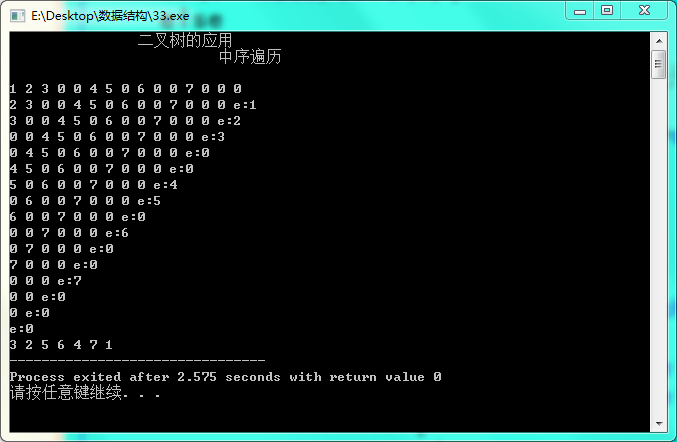
1. 判断是否为二叉排序树。

判断一个结点：与左孩子比较，与右孩子比较，递归调用，前提条件是左右孩子存在。

1. 测试数据和结果

注:上面是显示文件传递是否正确





5、算法时间复杂度

时间复杂度基本是0（n）；

改进：能够用图像界面显示大致图像(困难)

1. 源代码

#include<iostream>

using namespace std;

#include<malloc.h>

#include<fstream>

#define max 100

#define add 10

typedef struct BiTNode//二叉链表结构体

{

int data;

struct BiTNode \*lchild,\*rchild;//左右孩子指针

}BiTNode,\*BiTree;

typedef struct QNode//队列

{

BiTNode \*data;//（int data存数据BITNode\* data存树的指针）

struct QNode \*next;//下一个数据的指针

}QNode,\*QueuePtr;

typedef struct

{

QueuePtr front;//队列头

QueuePtr rear;//队列尾

}LinkQueue;

typedef struct//栈的结构体

{

BiTNode \* \*base;//栈底指针

BiTNode \* \*top;//栈顶指针

int stacksize;//栈的大小

}Sqstack;

void Init\_Queue(LinkQueue &s);//创建队列

void EnQueue(LinkQueue &s,BiTNode \*e);//入队

void DeQueue(LinkQueue &s,BiTNode \* &e);//出队

int Empty\_Queue(LinkQueue s);//判断队是否为空

void Init\_Sqstack(Sqstack &s);//构建栈

void Push(Sqstack &s,BiTNode \*e);//入栈

void Pop(Sqstack &s,BiTNode\* &e);//出栈

int Empty\_stack(Sqstack s);//判断栈是否为空

void Gottop\_stack(Sqstack s,BiTNode\* &e);//得到栈顶元素

void Init\_BiTree(BiTree &T);//创建二叉树

void Creat\_BiTree(BiTree &T);//创二叉树建数据

void Clear\_BiTree(BiTree &T);//清空树

bool Empty\_BiTree(BiTree T);//判断树是否为空

int Max(int x,int y);//简易的两个数比较大小

int Depth\_BiTree(BiTree T);//树的深度或者高度

void Preorder\_BiTree(BiTree T);//先序遍历

void Preorder\_\_BiTree(BiTree T);//先序遍历非递归

void Inorder\_BiTree(BiTree T);//中序遍历

void Inorder\_\_BiTree(BiTree T);//中序遍历非递归

void Postorder\_BiTree(BiTree T);//后序遍历

void Postorder\_\_BiTree(BiTree T);//后序遍历非递归

void div\_BiTree(BiTree T);//层次遍历

int Leaf\_BiTree(BiTree T);//求叶子数即宽度

int Paixu\_BiTree(BiTree T);//判断是否为二叉排序树

void CreatWen\_BiTree();//创建存树文件

void CreatHWen\_BiTree();//从文件传值给树

void Init\_Queue(LinkQueue &s)//创建队列

{

s.front=s.rear=(QueuePtr)malloc(sizeof(QNode));

if(!s.front)exit(0);

s.front->next=NULL;

}

void EnQueue(LinkQueue &s,BiTNode \*e)//入队

{

QueuePtr p;

p=(QueuePtr)malloc(sizeof(QNode));//开辟一个对应空间存数据

if(!p)exit(0);

p->data=e;

p->next=NULL;

s.rear->next=p;//可以看出队头为空，队尾不空

s.rear=p;

}

void DeQueue(LinkQueue &s,BiTNode \* &e)//出队

{

if(s.rear==s.front)exit(0);

QueuePtr p;

p=s.front->next;

e=p->data;

s.front->next=p->next;

if(s.rear==p)s.rear=s.front;

free(p);//删除空间

}

int Empty\_Queue(LinkQueue s)//判断队是否为空

{

if(s.rear==s.front)

return 1;

else

return 0;

}

void Init\_Sqstack(Sqstack &s)//构建栈

{

s.base=(BiTNode \*\*)malloc(max\*sizeof(BiTNode\*));

if(!s.base)exit(0);//如果失败，关闭

s.top=s.base;

s.stacksize=max;

}

void Push(Sqstack &s,BiTNode \*e)//入栈

{//s.top-s.base为栈的长度

if(s.top-s.base>=s.stacksize)//如果栈满

{

s.base=(BiTNode \*\*) realloc (s.base,(s.stacksize+add)\*sizeof( BiTNode\*));

if( !s.base)exit(0);

s.top=s.base+s.stacksize;

s.stacksize+=add;

}

\*s.top=e;//栈顶赋值

s.top++;//栈顶指针下移

}

void Pop(Sqstack &s,BiTNode\* &e)//出栈

{

if(s.base==s.top)exit(0);//如果栈空，关闭

s.top--;//因为栈顶为空，所以栈顶指针下移即出栈

e=\*s.top;

//e 赋值 栈当前值 即出栈的数据或者指针

}

int Empty\_stack(Sqstack s)//判断栈是否为空

{

if(s.base==s.top)

return 1;

else

return 0;

}

void Gottop\_stack(Sqstack s,BiTNode\* &e)//得到栈顶元素

{

if(s.base==s.top)exit(0);

s.top--;

//只需栈顶元素下移

e=\*s.top;

}

void Init\_BiTree(BiTree &T)//创建二叉树

{

T=(BiTNode \*)malloc(sizeof(BiTNode));

if(!T)exit(0);

T->lchild=T->rchild=NULL;//左右孩子置空

}

void Creat\_BiTree(BiTree &T)//创二叉树建数据

{//根据先序遍历规则来建树

int e;

fstream file,file1;

BiTNode f;

file.open("tree1.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

file1.open("tree2.dat",ios::out|ios::binary);

if(file1.fail())

{

cout<<"file1 open error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

e=f.data;

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

file1.write((char \*)&f,sizeof(f));

file.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

file.open("tree1.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

file1.open("tree2.dat",ios::in|ios::binary);

if(file1.fail())

{

cout<<"file1 open error!"<<endl;

exit(0);

}

file1.read((char \*)&f,sizeof(f));

while(!file1.eof())

{

file.write((char \*)&f,sizeof(f));

file1.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

cout<<endl;

file.open("tree1.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<f.data<<" ";

file.read((char \*)&f,sizeof(f));

}

file.close();

cout<<"e:"<<e<<" ";

if(e==0)

{

T=NULL;

}

else

{

T=(BiTNode \*)malloc(sizeof(BiTNode));

if(!T)exit(0);

T->data=e;

Creat\_BiTree(T->lchild);

Creat\_BiTree(T->rchild);

}

}

void Clear\_BiTree(BiTree &T)//清空树

{

if(T)

{

Clear\_BiTree(T->lchild);

Clear\_BiTree(T->rchild);

free(T);

T=NULL;

}

}

bool Empty\_BiTree(BiTree T)//判断树是否为空

{

if(!T)

return false;

}

int Max(int x,int y)//简易的两个数比较大小

{

if(x>y)

return x;

else

return y;

}

int Depth\_BiTree(BiTree T)//树的深度或者高度

{

if(T==NULL)

return 0;

else

return(Max(Depth\_BiTree(T->lchild),Depth\_BiTree(T->rchild))+1);

}

void Preorder\_BiTree(BiTree T)//先序遍历

{

if(T)

{

cout<<T->data<<" ";//遍历根

Preorder\_BiTree(T->lchild);//遍历左子树

Preorder\_BiTree(T->rchild);//遍历右子树

}

}

void Preorder\_\_BiTree(BiTree T)//先序遍历非递归

{//用栈实现

//思想：根入栈，出栈 右子树 左子树入栈

//即保证出栈的顺序是按照根 左子树 右子树。

Sqstack s;

BiTNode \*p;//树的指针

Init\_Sqstack(s);//创建一个栈

if(T)

Push(s,T);//如果树不空，根入栈

while(!Empty\_stack(s))//栈不空为循环条件

{

Pop(s,p);

cout<<p->data<<" ";

if(p->rchild)//如果右孩子不空入栈

{

Push(s,p->rchild);

}

if(p->lchild)//如果左孩子不空入栈

{

Push(s,p->lchild);

}

}

}

void Inorder\_BiTree(BiTree T)//中序遍历

{

if(T)

{

Inorder\_BiTree(T->lchild);//遍历左子树

cout<<T->data<<" ";//遍历根

Inorder\_BiTree(T->rchild);//遍历 右子树

}

}

void Inorder\_\_BiTree(BiTree T)//中序遍历非递归

{

//思想：不断的将根 左子树入栈，出栈时，右子树入栈

//即保证出栈的顺序是按照左子树 根 右子树。

BiTNode \*p=T;

Sqstack s;//建栈

Init\_Sqstack(s);

while(p||!Empty\_stack(s))

{

if(p)

{

Push(s,p);//根入栈

p=p->lchild;//指针移到左孩子

}

else

{

Pop(s,p);//出栈

cout<<p->data<<" ";

p=p->rchild;//右孩子入栈

}

}

}

void Postorder\_BiTree(BiTree T)//后序遍历

{

if(T)

{

Postorder\_BiTree(T->lchild);//遍历左子树

Postorder\_BiTree(T->rchild);//遍历右子树

cout<<T->data<<" ";//遍历根

}

}

void Postorder\_\_BiTree(BiTree T)//后序遍历非递归

{

//思想：控制出栈的条件是左子树的左右孩子为空，或者是已经访问过

BiTNode \*p=T;

Sqstack s;

Init\_Sqstack(s);//建栈

int word=1;

if(p->rchild)

{

Push(s,p->rchild);//右孩子入栈

}

Push(s,p);//根入栈

if(p->lchild)//左孩子入栈

{

Push(s,p->lchild);

}//先存入第一个右孩子和左孩子，根

Gottop\_stack(s,p);//得到栈顶元素，即左孩子

while(p)//得到左孩子

{

if(p->rchild&&p->rchild->data!='0')

{

Push(s,p->rchild);//右孩子入栈

}

if(p->lchild&&p->lchild->data!='0')

{

Push(s,p->lchild);//左孩子入栈

}

//注：根是原来的左孩子已经不需要入栈

if((p->lchild==NULL||p->lchild->data=='0')&&(p->rchild==NULL||p->rchild->data=='0'))

{

Pop(s,p);//出栈

cout<<p->data<<" ";

T=p;

T->data='0';//将树的数据暂时置‘0’

}

Gottop\_stack(s,p);//得到栈顶元素

}

}

void div\_BiTree(BiTree T)//层次遍历

{

LinkQueue s;

BiTNode \*p=T;

Init\_Queue(s);//建队

if(T)

{

EnQueue(s,p);//入队

}

while(!Empty\_Queue(s))

{

DeQueue(s,p);//出队

cout<<p->data<<" ";

if(p->lchild)

EnQueue(s,p->lchild);//左孩子入队

if(p->rchild)

EnQueue(s,p->rchild);//右孩子入队

}

}

int Leaf\_BiTree(BiTree T)//求叶子数即宽度

{

if(T==NULL)

return 0;

else

{

if(T->lchild==NULL&&T->rchild==NULL)

return 1;

else

return(Leaf\_BiTree(T->lchild)+(Leaf\_BiTree(T->rchild)));

}

}

int Paixu\_BiTree(BiTree T)//判断是否为二叉排序树

{

//思想：通过比较根与左孩子 根与右孩子的情况

if(T==NULL)exit(0);

else

{

if(T->lchild)//如果存在左孩子

{

if(T->data < T->lchild->data)

return 0;

else

{

if(T->rchild)

{

if(T->data<T->rchild->data)

{

Paixu\_BiTree(T->lchild);

Paixu\_BiTree(T->rchild);

}

else

return 0;

}

else

Paixu\_BiTree(T->lchild);

}

}

else//如果不存在左孩子

{

if(T->rchild)

if(T->data>T->rchild->data)

{

Paixu\_BiTree(T->rchild);

}

else

return 0;

}

return 1;

}

}

void CreatWen\_BiTree()//创建存树文件

{

fstream file;

file.open("tree.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

BiTNode f;

int key=1;

cout<<"please input the all data(0->NULL -1->end):"<<endl;

while(key)

{

int elem;

cin>>elem;

f.data=elem;

f.lchild=NULL;

f.rchild=NULL;

if(elem==-1)

{

key=0;

}

else

file.write((char \*)&f,sizeof(f));

}

file.close();

file.open("tree.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<f.data<<" ";

file.read((char \*)&f,sizeof(f));

}

file.close();

cout<<endl;

}

void CreatHWen\_BiTree()

{

fstream file,file1;

BiTNode f;

file.open("tree.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

file1.open("tree1.dat",ios::out|ios::binary);

if(file1.fail())

{

cout<<"file1 open error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

file1.write((char \*)&f,sizeof(f));

file.read((char \*)&f,sizeof(f));

}

file.close();

file1.close();

cout<<endl;

file.open("tree1.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<f.data<<" ";

file.read((char \*)&f,sizeof(f));

}

file.close();

}

int main()

{

int word;

cout<<" 二叉树的应用"<<endl;

cout<<" 0.创建文件"<<endl;

cout<<" 1.构建二叉树 "<<endl;

cout<<" 2.求树的高度"<<endl;

cout<<" 3.求树的深度"<<endl;

cout<<" 4.先序遍历"<<endl;

cout<<" 5.中序遍历"<<endl;

cout<<" 6.后序遍历"<<endl;

cout<<" 7.先序遍历非递归"<<endl;

cout<<" 8.中序遍历非递归"<<endl;

cout<<" 9.后序遍历非递归"<<endl;

cout<<" 10.判断是否为二叉排序树"<<endl;

cout<<"please input your choose:";

cin>>word;

switch(word)

{

case 0:

{

system("cls");

cout<<" 二叉树的应用"<<endl;

cout<<" 创建文件"<<endl;

CreatWen\_BiTree();

CreatHWen\_BiTree();

break;

}

case 1:

{

system("cls");

cout<<" 二叉树的应用"<<endl;

cout<<" 构建二叉树 "<<endl;

CreatHWen\_BiTree();

BiTree T;

Init\_BiTree(T);//构建二叉树

Creat\_BiTree(T);//初始化二叉树

cout<<endl;

Preorder\_BiTree(T);//先序遍历 \*/

break;

}

case 2:

{

system("cls");

cout<<" 二叉树的应用"<<endl;

cout<<" 求树的高度"<<endl;

CreatHWen\_BiTree();

BiTree T;

Init\_BiTree(T);//构建二叉树

Creat\_BiTree(T);//初始化二叉树

cout<<endl;

cout<<"Depth:"<<Depth\_BiTree(T)<<endl;

break;

}

case 3:

{

system("cls");

cout<<" 二叉树的应用"<<endl;

cout<<" 求树的深度"<<endl;

CreatHWen\_BiTree();

BiTree T;

Init\_BiTree(T);//构建二叉树

Creat\_BiTree(T);//初始化二叉树

cout<<endl;

cout<<"deepth:"<<Leaf\_BiTree(T)<<endl;

break;

}

case 4:

{

system("cls");

cout<<" 二叉树的应用"<<endl;

cout<<" 先序遍历"<<endl;

CreatHWen\_BiTree();

BiTree T;

Init\_BiTree(T);//构建二叉树

Creat\_BiTree(T);//初始化二叉树

cout<<endl;

Preorder\_BiTree(T);//先序遍历

break;

}

case 5:

{

system("cls");

cout<<" 二叉树的应用"<<endl;

cout<<" 中序遍历"<<endl;

CreatHWen\_BiTree();

BiTree T;

Init\_BiTree(T);//构建二叉树

Creat\_BiTree(T);//初始化二叉树

cout<<endl;

Inorder\_BiTree(T);//中序遍历

break;

}

case 6:

{

system("cls");

cout<<" 二叉树的应用"<<endl;

cout<<" 后序遍历"<<endl;

CreatHWen\_BiTree();

BiTree T;

Init\_BiTree(T);//构建二叉树

Creat\_BiTree(T);//初始化二叉树

cout<<endl;

Postorder\_BiTree(T);//后序遍历

break;

}

case 7:

{

system("cls");

cout<<" 二叉树的应用"<<endl;

cout<<" 先序遍历非递归"<<endl;

CreatHWen\_BiTree();

BiTree T;

Init\_BiTree(T);//构建二叉树

Creat\_BiTree(T);//初始化二叉树

cout<<endl;

Preorder\_\_BiTree(T);//先序遍历

break;

}

case 8:

{

system("cls");

cout<<" 二叉树的应用"<<endl;

cout<<" 中序遍历非递归"<<endl;

CreatHWen\_BiTree();

BiTree T;

Init\_BiTree(T);//构建二叉树

Creat\_BiTree(T);//初始化二叉树

cout<<endl;

Inorder\_\_BiTree(T);//中序遍历

break;

}

case 9:

{

system("cls");

cout<<" 二叉树的应用"<<endl;

cout<<" 后序遍历非递归"<<endl;

CreatHWen\_BiTree();

BiTree T;

Init\_BiTree(T);//构建二叉树

Creat\_BiTree(T);//初始化二叉树

cout<<endl;

Postorder\_\_BiTree(T);//后序遍历

break;

}

case 10:

{

system("cls");

cout<<" 二叉树的应用"<<endl;

cout<<" 判断是否为二叉排序树"<<endl;

CreatHWen\_BiTree();

BiTree T;

Init\_BiTree(T);//构建二叉树

Creat\_BiTree(T);//初始化二叉树

cout<<endl;

cout<<"1(true)or 0(false):"<<Paixu\_BiTree(T)<<endl;

break;

}

default:

cout<<"input data is error!";

}

return 0;

}

四、Huffman编码与解码

1、题目简介

Huffman编码与解码(必做)（Huffman编码、二叉树）

[问题描述]

对一篇不少于2000字符的英文文章，统计各字符出现的次数，实现Huffman编码，以及对编码结果的解码。

[基本要求]

（1） 输出每个字符出现的次数和编码。

（2） 在Huffman编码后，要将编码表和英文文章编码结果保存到文件中，编码结果必须是二进制形式，即 0 1的信息用比特位表示，不能用字符’0’和’1’表示。

（3） 提供读编码文件生成原文件的功能。

2、数据结构

#define MAX 10000

typedef struct Eglish//存文章的字符 及次数

{

char ascall[128];//字符

int weight[128];//次数

int n;

int M;//开辟数组的长度

}English;

//静态三叉链表存储结构

typedef struct//编码表

{

int weight;//权值只能是正数

int parent;

int lchild;

int rchild;

}HTNode;

typedef struct//文件结构体

{

char ch;

int weight;//权值只能是正数

int parent;

int lchild;

int rchild;

}H;

typedef HTNode HuffmanTree[128];//0号单元不使用

typedef char \* HuffmanCode[128];//存储每个字符的哈夫曼编码表

//是一个字符指针数组，每个数组元素是指向字符指针的指针

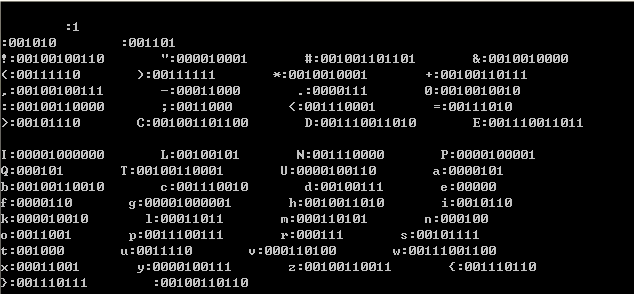
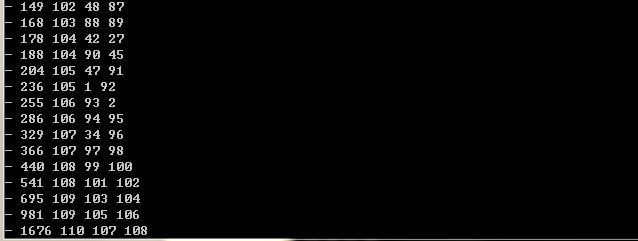
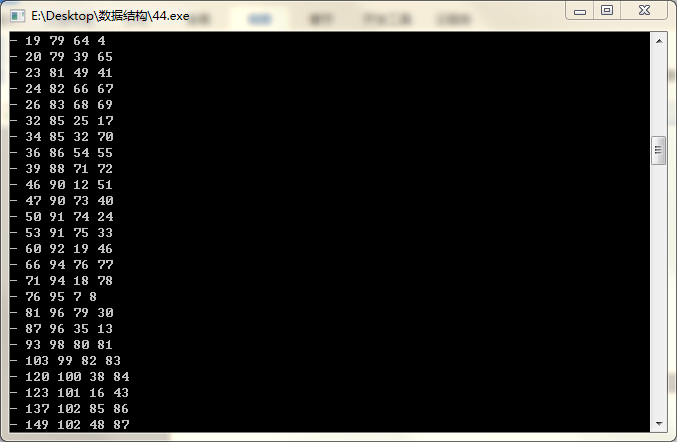
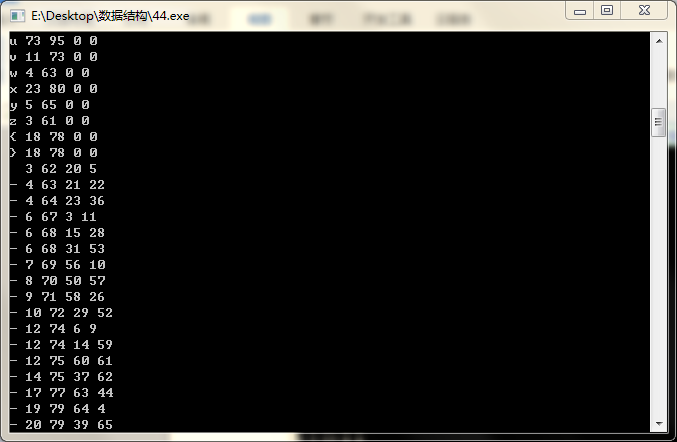
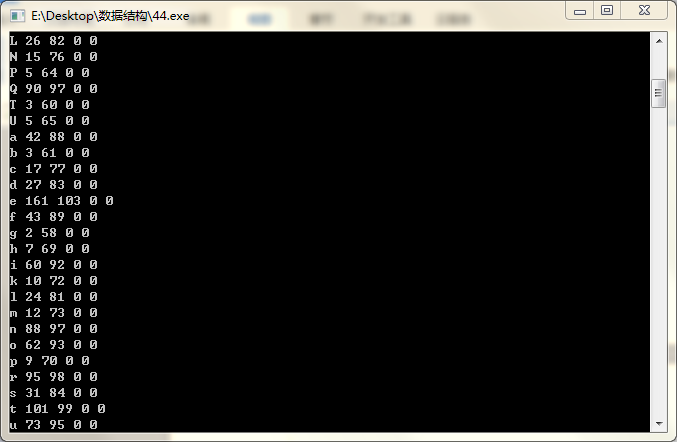
typedef char \* HuffmanCode[128];//存储每个字符的哈夫曼编码表

1. 算法设计思想

建立Huffman编码表。（我的理解是模拟树的结构）基本思想就是如书上。选两个最小结点，然后建树。

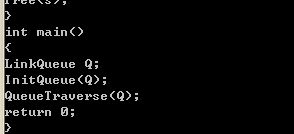
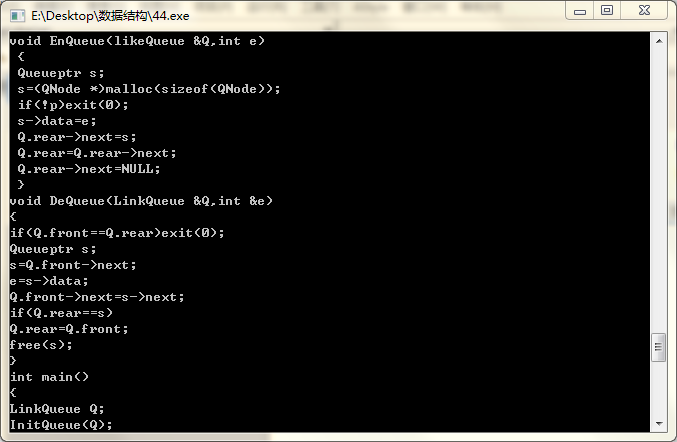
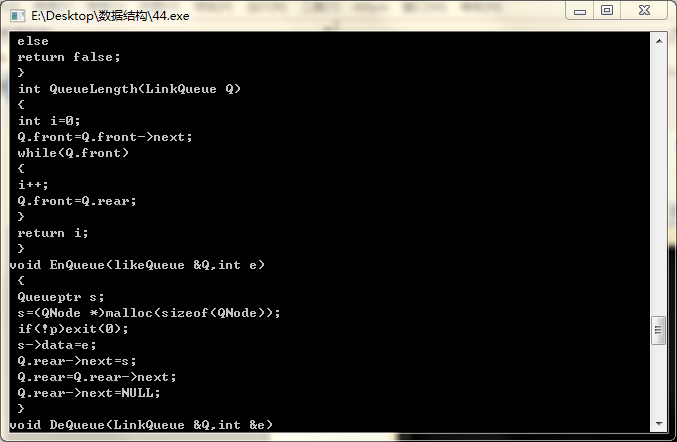
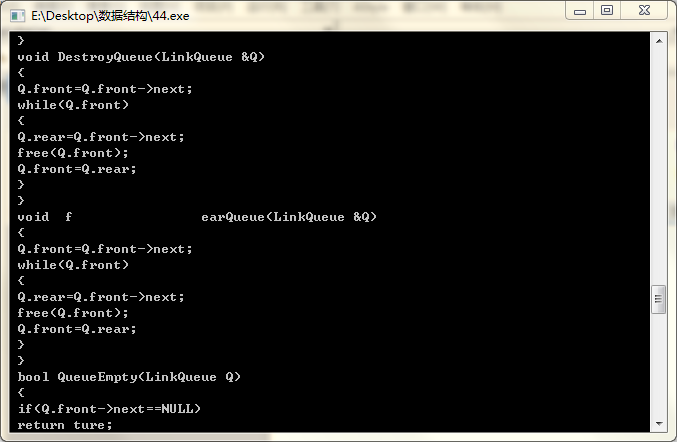
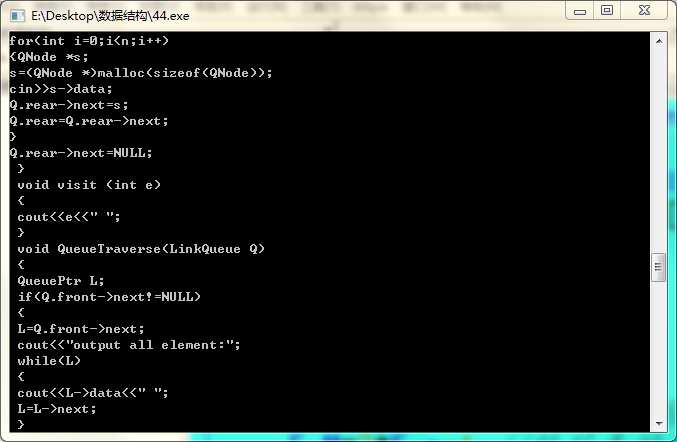
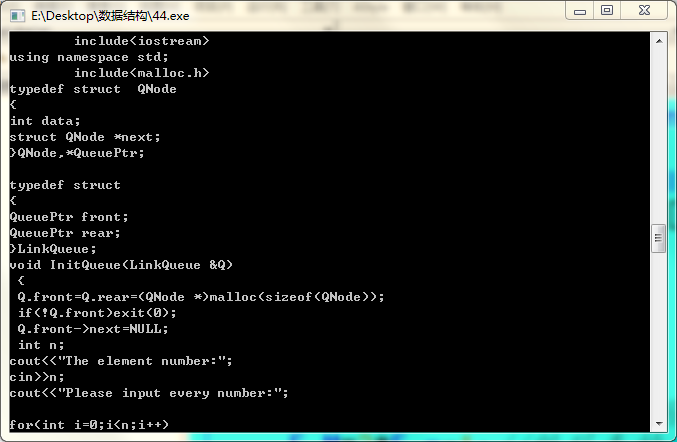
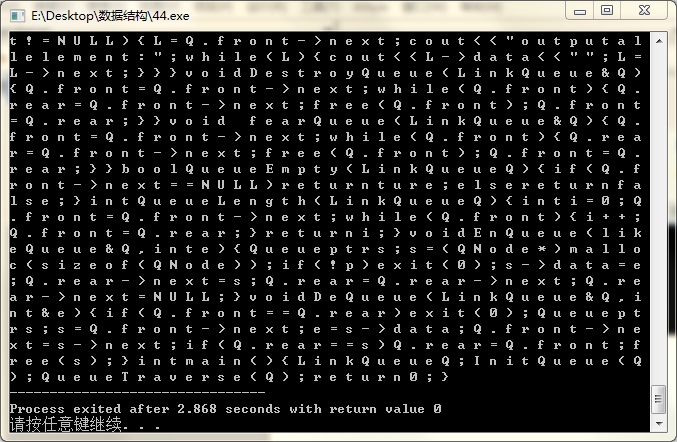
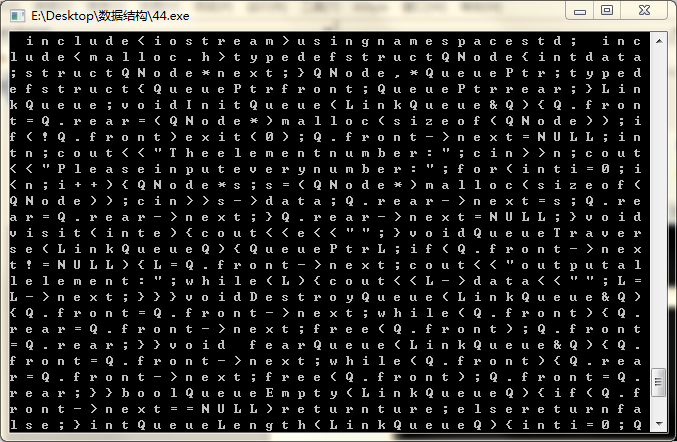
编码：从结点位置不断找到根结点，并存入临时数组。（存法：倒着存），然后又传递给编码数组；

解码:与编码相反。

1. 测试数据和结果

编码文章：

解码文章：

1. 数组存显示
2. 文件存显示

5、算法时间复杂度

时间复杂度基本是0（n）；

6、源代码

#include<iostream>

using namespace std;

#include<fstream>

#include<malloc.h>

#include<stdlib.h>

#include<string.h>

#define MAX 10000

typedef struct Eglish//存文章的字符 及次数

{

char ascall[128];//字符

int weight[128];//次数

int n;

int M;//开辟数组的长度

}English;

//静态三叉链表存储结构

typedef struct//编码表

{

int weight;//权值只能是正数

int parent;

int lchild;

int rchild;

}HTNode;

typedef struct//文件结构体

{

char ch;

int weight;//权值只能是正数

int parent;

int lchild;

int rchild;

}H;

typedef HTNode HuffmanTree[128];//0号单元不使用

typedef char \* HuffmanCode[128];//存储每个字符的哈夫曼编码表，

//是一个字符指针数组，每个数组元素是指向字符指针的指针

typedef char \* HuffmanCode[128];//存储每个字符的哈夫曼编码表

void Creat\_wen(English &E);//创建文件

void select(HuffmanTree HT, int k, int &s1, int &s2);//k代表表当前位置前面s1代表最小值s2表次小值

void Create\_HuffmanTree(HuffmanTree &HT,Eglish E);//创建编码表

void open\_code();//open哈夫曼编码表

void Decode\_HuffmanCode(HuffmanTree HT,Eglish E, char Str[], int len, char result[]);//解码

void Print\_HuffmanTree(HuffmanTree HT, Eglish E);//打印编码表

void Put(HuffmanCode HC,Eglish E);//将英文文章翻译成编码文章

void Get(HuffmanTree HT,English E);//将编码文章翻译英文文章

void Print\_HuffmanCode(HuffmanCode HC, Eglish E);//每个字符的编码

void Encode\_HuffmanCode(HuffmanTree HT,Eglish E ,HuffmanCode &HC);//编码 加密

void Creat\_wen(English &E)//创建文件

{

fstream file;

int Ascall[128]={0};

file.open("haff.txt",ios::in);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

char ch;

file.get(ch);

while(!file.eof())

{

Ascall[ch]++;

file.get(ch);

}

file.close();

E.n=0;

for(int i=0;i<128;i++)//筛选需要的字符

{

if(Ascall[i]>0)

{

E.ascall[E.n]=i;

E.weight[E.n]=Ascall[i];

E.n++;

}

}

for(int i=0;i<E.n;i++)

{

if(i!=0&&(i%10==0))

cout<<endl;

cout<<E.ascall[i]<<":"<<E.weight[i]<<" ";

}

cout<<endl;

}

//parent不为0说明该结点已经参与构造了，故不许再考虑

void select(HuffmanTree HT, int k, int &s1, int &s2)//k代表表当前位置前面s1代表最小值s2表次小值

{

//假设s1对应的权值总是<=s2对应的权值

int tmp = MAX, word = 0;//tmp用于找到最小值，word用于标记位置

for(int i = 1; i <= k; i++)

{

if(!HT[i].parent)//parent必须为0

{

if(tmp > HT[i].weight)

{

tmp = HT[i].weight;//tmp最后为最小的weight

word = i;

}

}

}

s1 = word;

tmp = MAX;

int key = 0;//tmp用于找到次小值，key用于标记位置

for(int i = 1; i <= k; i++)

{

if((!HT[i].parent) && i!=s1)//parent必须为0和不能等于最小值

{

if(tmp > HT[i].weight)

{

tmp = HT[i].weight;

key = i;

}

}

}

s2 = key;

}

//构造哈夫曼树

//void Create\_HuffmanTree(HuffmanTree &HT, int \*w, int n)

void Create\_HuffmanTree(HuffmanTree &HT,Eglish E)

{

if(E.n <=0)

return;

//对树赋初值

for(int i = 0; i <=E.n; i++) {

HT[i].weight = E.weight[i];

HT[i].lchild = 0;

HT[i].parent = 0;

HT[i].rchild = 0;

}

int M=2\*E.n-1;

for(int i=E.n; i <=E.M; i++)

{

HT[i].weight = 0;

HT[i].lchild = 0;

HT[i].parent = 0;

HT[i].rchild = 0;

}

for(int i =E.n; i <=E.M; i++)

{

int s1, s2;

select(HT, i-1, s1, s2);

HT[s1].parent = i;

HT[s2].parent = i;

HT[i].lchild = s1;

HT[i].rchild = s2;

HT[i].weight = HT[s1].weight + HT[s2].weight;

}

fstream file;

file.open("code.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

H f;

for(int i = 0; i <= E.M; i++)

{

if(i > E.n)

f.ch='-';

else

f.ch=E.ascall[i];

f.lchild=HT[i].lchild;

f.parent=HT[i].parent;

f.rchild=HT[i].rchild;

f.weight=HT[i].weight;

file.write((char \*)&f,sizeof(f));

}

file.close();

}

void open\_code()//open哈夫曼编码表，

{

fstream file;

file.open("code.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

H f;

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<f.ch <<" "<<f.weight<<" "<<f.parent<<" "<<f.lchild<<" "<<f.rchild<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

}

void Decode\_HuffmanCode(HuffmanTree HT,Eglish E, char Str[], int len, char result[])//解码

{ fstream file1;

file1.open("haff2.txt",ios::out);

if(file1.fail())

{

cout<<"file1 open error!"<<endl;

exit(0);

}

int p =E.M;//HT的最后一个节点是根节点，前n个节点是叶子节点

int i = 0;//指示测试串中的第i个字符

//char result[30];//存储解码以后的字符串

int j = 0;//指示结果串中的第j个字符

while(i<len)

{

if(Str[i] == '0')

{

p = HT[p].lchild;

}

if(Str[i] == '1')

{

p = HT[p].rchild;

}

if(p <= E.n)

{//p<=N则表明p为叶子节点,因为在构造哈夫曼树HT时，HT的m个节点中前n个节点为叶子节点

result[j] = E.ascall[p];

cout<<result[j];

file1<<E.ascall[p]<<" ";

j++;

p = E.M;//p重新指向根节点

}

i++;

}

cout<<endl;

cout<<"j:"<<j<<endl;

// result[j] = '\0';//结果串的结束符

file1.close();

}

void Print\_HuffmanTree(HuffmanTree HT, Eglish E)//打印编码表

{

cout<<endl;

cout<<"data, weight, parent, lchild, rchild"<<endl;

for(int i = 0; i <= E.M; i++){

if(i > E.n)

{

cout<<"-"<<" "<<HT[i].weight<<" "<<HT[i].parent<<" "<<HT[i].lchild<<" "<<HT[i].rchild<<endl;

}

else

{

cout<<E.ascall[i]<<" "<<HT[i].weight<<" "<<HT[i].parent<<" "<<HT[i].lchild<<" "<<HT[i].rchild<<endl;

}

}

}

void Put(HuffmanCode HC,Eglish E)//将英文文章翻译成编码文章

{

fstream file,file1;

file.open("haff.txt",ios::in);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

file1.open("haff1.txt",ios::out);

if(file1.fail())

{

cout<<"file1 open error!"<<endl;

exit(0);

}

char ch;

file.get(ch);

while(!file.eof())

{

//cout<<HC[E.ascall[ch]]<<" ";

for(int i=1;i<=E.n;i++)

{

if(E.ascall[i]==ch)

{

file1<<HC[i];

break;

}

}

file.get(ch);

}

file.close();

file1.close();

file.open("haff1.txt",ios::in);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

char c;

file.get(c);

while(!file.eof())

{

cout<<c;

file.get(c);

}

file.close();

}

void Get(HuffmanTree HT,English E)//将编码文章翻译英文文章

{

char str[1000000];

fstream file,file1;

file.open("haff1.txt",ios::in);

if(file.fail())

{

cout<<"file open error!"<<endl;

exit(0);

}

char ch;

file>>ch;

int i=0;

while(!file.eof())

{

str[i]=ch;

i++;

file>>ch;

}

file.close();

str[i]='\0';

char result[1000001];

cout<<endl;

/\* cout<<"222"<<endl;

for(int j=0;j<i;j++)

{

cout<<str[j];

}\*/

Decode\_HuffmanCode(HT, E,str, i, result);

file1.open("haff2.txt",ios::in);

if(file1.fail())

{

cout<<"file1 open error!"<<endl;

exit(0);

}

file1>>ch;

while(!file1.eof())

{

cout<<ch<<" ";

file1>>ch;

}

for(int j=0;result[j]!='\0';j++)

{

cout<<result[j];

}

file1.close();

}

void Print\_HuffmanCode(HuffmanCode HC, Eglish E)//每个字符的编码

{

cout<<endl;

for(int t=1,i = 0; i <=E.n; t++,i++)

{

if(t%4==0)

cout<<endl;

cout<<E.ascall[i]<<":"<<HC[i]<<" ";

}

cout<<endl;

}

void Encode\_HuffmanCode(HuffmanTree HT,Eglish E ,HuffmanCode &HC)

{

//每个字符的编码长度不会超过n

char tmp[E.n];

tmp[E.n-1] = '\0';//编码的结束符

int start, c, f;

for(int i = 0; i <=E.n; i++)

{//对于第i个待编码字符即第i个带权值的叶子节点

start =E.n-1;//编码生成以后，start将指向编码的起始位置

c = i;

f = HT[i].parent;

while(f)

{//f!=0,即f不是根节点的父节点

if(HT[f].lchild == c)

{

tmp[--start] = '0';

}

else

{//HT[f].rchild == c,注意:由于哈夫曼树中只存在叶子节点和度为2的节点，

//所以除开叶子节点，节点一定有左右2个分支

tmp[--start] = '1';

}

c = f;

f = HT[f].parent;

}

HC[i] = (char \*)malloc((E.n-start)\*sizeof(char));//每次tmp的后n-start个位置有编码存在

strcpy(HC[i], &tmp[start]);//将tmp的后n-start个元素分给H[i]指向的的字符串

}

}

int main()

{

English E;

Creat\_wen(E);//创建编码数组 即从文件中读出字符的及其次数

E.M=2\*E.n-1;//编码表的长度

HuffmanTree HT;

Create\_HuffmanTree(HT,E);

open\_code();//编码表

//Print\_HuffmanTree(HT,E);//输出编码表

HuffmanCode HC;//HC有n个元素，每个元素是一个指向字符串的指针，

//即每个元素是一个char \*的变量

Encode\_HuffmanCode(HT,E, HC);//为每个字符求解哈夫曼编码

Print\_HuffmanCode(HC,E);

Put(HC,E);

Get(HT,E);

}

五、无线传感器网络

1、题目简介

[问题描述]

在一个直角坐标（设为100\*100）平面中，随机分布n无线传感器结点，汇聚点为原点（0，0），要求将每个无线传感器结点的信息传输到汇聚点，传输可以通过多跳方式实现，设无线传感器结点最大无线通信距离为10，传输能耗与距离平方成正比，传输时间与距离成正比。

问题一：建立每个结点传输到汇聚点的最短时间通道，并找出无法实现传输的结点，将其排除。

问题二：建立整个网络传输到汇聚点的平均能耗最小网络结构。

[基本要求]

输入格式： 输入的第一行包含一个正整数n，表示无线传感器结点数量。结点使用1, 2, 3, ……n依次标号。 接下来n行，每行包含三个整数ni, xi, yi，其中xi, yi表示第ni个结点的坐标，要求从文本文件中输入。

输出格式：

问题一：输出每个结点到汇聚点的最短时间通道，包括最短时间和经过结点。

问题二：输出整个网络到汇聚点的平均能耗最小网络结构，包括平均最小能耗和连接方式。

1. 数据结构

#define Infine 10000

#define max 100

点：

typedef struct

{

int node;//结点

int i;//坐标

int j;//坐标

}node;

图：邻接矩阵

typedef struct

{

int \*ver;//顶点

int arc[max+1][max+1];//边

int Ver,Edg;//顶点数和边数

}Graph;

栈：

#define STACK\_INIT\_SIZE 100

#define STACKINCREAT 10

typedef struct

{

int \*base;

int \*top;

int stacksize;

}SqStack;

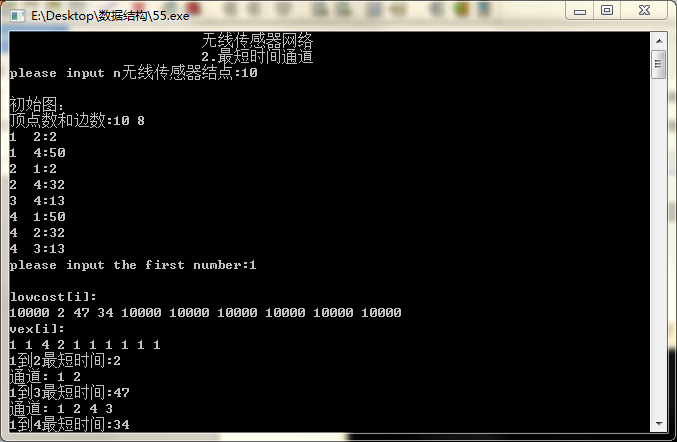
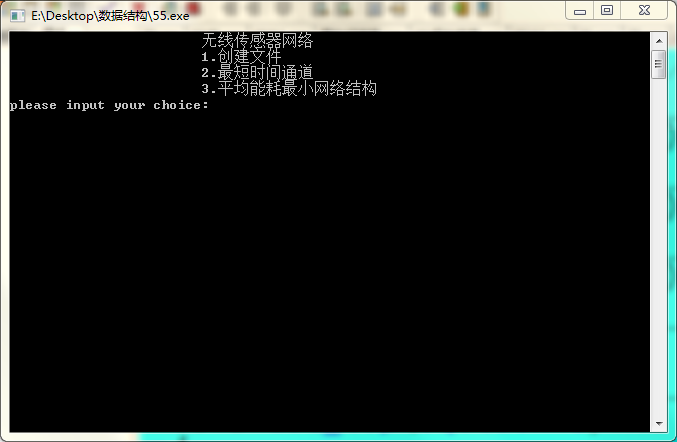
1. 算法设计思想

注：结点的筛选非常严谨：无线传感器结点最大无线通信距离为10，找出无法实现传输的结点，将其排除。

所以在创建文件时，结点数并不代表真正的图，需要筛选出可行的图，也是这题的难点。

问题一：输出每个结点到汇聚点的最短时间通道，包括最短时间和经过结点。 （最短路径） 依照prim算法改编

问题二：输出整个网络到汇聚点的平均能耗最小网络结构，包括平均最小能耗和连接方式。（最小生成树）prim算法

4、测试数据和结果

5、算法时间复杂度

时间复杂度基本是0（n）；

改进想法：图的显示更加明显

6、源代码

#include<iostream>

using namespace std;

#include<malloc.h>

#include<fstream>

#include<math.h>

#define Infine 10000

#define max 100

typedef struct

{

int node;

int i;

int j;

}node;

typedef struct

{

int \*ver;//顶点

int arc[max+1][max+1];//边

int Ver,Edg;//顶点数和边数

}Graph;

#define STACK\_INIT\_SIZE 100

#define STACKINCREAT 10

typedef struct

{

int \*base;

int \*top;

int stacksize;

}SqStack;

void Init\_Graph(Graph &g);

void Creat\_map();//创建文件

void Creat\_Graph(Graph &g);//创建问题一的图

void Creat1\_Graph(Graph &g);//创建问题二的图

void Print\_Graph(Graph g);//打印图

void InitStack(SqStack &s);//构造空栈和赋值

void Print\_Stack(SqStack s);//打印栈

void Push(SqStack &s,int e);//添加一个元素

int Empty\_Stack(SqStack s);

void Pop(SqStack &s,int &e);//删除一个元素

void Got\_top(SqStack s,int &e);//得到栈顶的元素

void Search(Graph g,int lowcost[],int vex[],int k);//找路径

void Short\_Path(Graph g,int k);//最短路径 思路仿prim算法

void Prim(Graph g,int k);//最小生成树prim算法

void Init\_Graph(Graph &g)

{

g.ver=(int \*)malloc(max\*sizeof(int));

if(!g.ver)exit(0);

/\* int m,n,i,j;

/\* cout<<"输入顶点数和边数:";

cin>>m>>n;

g.Ver=m;//初始化顶点数

g.Edg=n;//初始化边数

for(i=1;i<=g.Ver;i++)//输入顶点信息标识:"

{

g.ver[i]=i;

}\*/

g.Edg=0;

g.Ver=0;

//把矩阵初始化为00

int i,j;

for(i=0;i<=max;i++)

for(j=0;j<=max;j++)

{

g.arc[i][j]=Infine;

}

}

void Creat\_map()//创建文件

{

fstream file;

file.open("net.txt",ios::out);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

int n;

cout<<"please input n无线传感器结点:";

cin>>n;

node nn[n];

//int ni,xi,yi;

cout<<"please input ni, xi, yi:"<<endl;

for(int i=1;i<=n;i++)

{

cin>>nn[i].node;//输入节点

cin>>nn[i].i;//输入节点位置

cin>>nn[i].j;//输入节点位置

file<<nn[i].node<<" ";

file<<nn[i].i<<" ";

file<<nn[i].j<<" ";

}

file.close();

cout<<"put file'content:"<<endl;

int k;

file.open("net.txt",ios::in);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file>>k;

while(!file.eof())

{

cout<<k<<" ";

file>>k;

}

file.close();

}

void Creat\_Graph(Graph &g)//创建问题一的图

{

int n;

cout<<"please input n无线传感器结点:";

cin>>n;

g.Ver=n;//节点数

node nn[n];

//int ni,xi,yi;

int count=0;

int k;

fstream file;

file.open("net.txt",ios::in);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file>>k;

for(int i=1;i<=n;i++)

{

nn[i].node=k;//输入节点

g.ver[i]=nn[i].node;

file>>k;

nn[i].i=k;//输入节点位置

file>>k;

nn[i].j=k;//输入节点位置

file>>k;

if(i>=2)//从第二个节点判断是否有边

{

for(int j=1;j<i;j++)//从第一个节点开始

{//distance求距离的平方

int distance=(nn[i].i-nn[j].i)\*(nn[i].i-nn[j].i)+(nn[i].j-nn[j].j)\*(nn[i].j-nn[j].j);

if(j==1&&sqrt(distance)>10)

{

break;

}

else if(sqrt(distance)<10)//如果距离小于10

{

count++;//记录边数

g.arc[nn[i].node][nn[j].node]=distance;//加边

g.arc[nn[j].node][nn[i].node]=distance;

}

}

}

}

file.close();

g.Edg=2\*count;//传递边数

/\* int i,j;

//输入有权值的边，其余直接赋值

for(int i=1;i<=g.Edg;i++)//输入顶点信息标识:"

{

int a,a1,weight,word=0;

cout<<"输入(vi,vj）顶点 权值:"<<endl;

cin>>a>>a1>>weight;////输入顶点 权值

for(int s=1;s<=g.Ver;s++)//判断是否顶点是否正确

{

if(g.ver[s]==a)

word++;

if(g.ver[s]==a1)

word++;

}

if(word==2)

{

g.arc[a][a1]=weight;

g.arc[a1][a]=weight;

}

else//如果不正确，重复输入

{

cout<<"input is error!";

i--;

}

} \*/

}

void Creat1\_Graph(Graph &g)//创建问题二的图

{

int n;

cout<<"please input n无线传感器结点:";

cin>>n;

g.Ver=n;//节点数

node nn[n];

//int ni,xi,yi;

int count=0;

int k;

fstream file;

file.open("net.txt",ios::in);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file>>k;

for(int i=1;i<=n;i++)

{

nn[i].node=k;//输入节点

g.ver[i]=nn[i].node;

file>>k;

nn[i].i=k;//输入节点位置

file>>k;

nn[i].j=k;//输入节点位置

file>>k;

if(i>=2)//从第二个节点判断是否有边

{

for(int j=1;j<i;j++)//从第一个节点开始

{//distance求距离的平方

int distance=(nn[i].i-nn[j].i)\*(nn[i].i-nn[j].i)+(nn[i].j-nn[j].j)\*(nn[i].j-nn[j].j);

if(j==1&&sqrt(distance)>10)

{

break;

}

else if(sqrt(distance)<10)//如果距离小于10

{

count++;//记录边数

g.arc[nn[i].node][nn[j].node]=sqrt(distance);//加边

g.arc[nn[j].node][nn[i].node]=sqrt(distance);

}

}

}

}

file.close();

g.Edg=2\*count;//传递边数

/\* int i,j;

//输入有权值的边，其余直接赋值

for(int i=1;i<=g.Edg;i++)//输入顶点信息标识:"

{

int a,a1,weight,word=0;

cout<<"输入(vi,vj）顶点 权值:"<<endl;

cin>>a>>a1>>weight;////输入顶点 权值

for(int s=1;s<=g.Ver;s++)//判断是否顶点是否正确

{

if(g.ver[s]==a)

word++;

if(g.ver[s]==a1)

word++;

}

if(word==2)

{

g.arc[a][a1]=weight;

g.arc[a1][a]=weight;

}

else//如果不正确，重复输入

{

cout<<"input is error!";

i--;

}

} \*/

}

void Print\_Graph(Graph g)//打印图

{

cout<<"顶点数和边数:"<<g.Ver<<" "<<g.Edg<<endl;

for(int i=0;i<=max;i++)

{

for(int s=0;s<=max;s++)

{

if(g.arc[i][s]!=Infine)

cout<<i<<" "<<s<<":"<<g.arc[i][s]<<endl;

}

}

}

void InitStack(SqStack &s)//构造空栈和赋值

{

s.base=(int \*)malloc(STACK\_INIT\_SIZE\*sizeof(int));

if(!s.base)exit(0);

s.top=s.base;

s.stacksize=STACK\_INIT\_SIZE;

}

void Print\_Stack(SqStack s)//打印栈

{

cout<<"the SqStack element:";

cout<<s.top-s.base<<endl;

SqStack s1;

s1.base=s.base;

for(int i=0; i<s.top-s.base; i++)//元素的长度length=s.top-s.base

{

cout<<\*s1.base++<<" ";

}

}

void Push(SqStack &s,int e)//添加一个元素

{

if(s.top-s.base>=s.stacksize)

{

s.base=(int \*)realloc(s.base,(s.stacksize+STACKINCREAT)\*sizeof(int));

if(!s.base)exit(0);

s.top=s.base+s.stacksize;

s.stacksize+=STACKINCREAT;

}

\*s.top=e;

s.top++;

}

void Pop(SqStack &s,int &e)//删除一个元素

{

if(s.base==s.top)exit(0);

--s.top;

e=\*s.top;

}

int Empty\_Stack(SqStack s)

{

if(s.base==s.top)

return 1;

else

return 0;

}

void Got\_top(SqStack s,int &e)//得到栈顶的元素

{

if(s.base==s.top)exit(0);

e=\*(s.top-1);

}

void Search(Graph g,int lowcost[],int vex[],int k)//找路径

{

int tongdao[g.Edg+1];

int s,t;

for(int ss=1;ss<=g.Edg;ss++)

{

tongdao[ss]=0;

}

for(int i=1;i<=g.Ver;i++)

{

s=1;

if(i!=k&&lowcost[i]!=Infine)

{

cout<<k<<"到"<<i<<"最短时间:"<<lowcost[i]<<endl;

tongdao[s]=i;//记录终点位置

t=i;//t记录终点位置

while(vex[t]!=k)

{

s++;

tongdao[s]=vex[t];

t=vex[t];

}

cout<<"通道: "<<k<<" ";

for(int j=s;j>=1;j--)

cout<<tongdao[j]<<" ";

cout<<endl;

}

}

}

void Short\_Path(Graph g,int k)//最短路径 思路仿prim算法

{

int m,n;

int visit[g.Ver+1];//访问标志

int lowcost[g.Ver+1];//权值

int vex[g.Ver+1];//存顶点

for(int i=1;i<=g.Ver;i++)

{

visit[i]=0;//访问标志

lowcost[i]=g.arc[k][i];//k距离i最小权值

vex[i]=k;

}

visit[k]=1;//k点被访问

int key=g.Ver-1;//循环次数

// cout<<"最小生成树生成边加入过程："<<endl;

while(key!=0)

{

int min;

int word;

for(int i=1;i<=g.Ver;i++)//设一个最小值

{

if(!visit[i])//未被访问的

{

min=lowcost[i];

word=i;

break;

}

}

for(int i=1;i<=g.Ver;i++)//找到权值最小值

{

if(!visit[i])

{

if(lowcost[i]<min)

{

min=lowcost[i];

word=i;//记录位置

}

}

}

visit[word]=1;//标记

key--;//次数减一

// cout<<"<"<<vex[word]<<","<<word<<">"<<" ";//输出下一个节点

for(int j=1;j<=g.Ver;j++)//刷新权值数组

{

if(!visit[j])

{

if(g.arc[word][j]+lowcost[word]<lowcost[j])

{

lowcost[j]=g.arc[word][j]+lowcost[word];//权值改变

vex[j]=word;//顶点数组改变

}

}

}

}

cout<<"lowcost[i]:"<<endl;

for(int i=1;i<=g.Ver;i++)

{

cout<<lowcost[i]<<" ";

}

cout<<endl;

cout<<"vex[i]:"<<endl;

for(int i=1;i<=g.Ver;i++)

{

cout<<vex[i]<<" ";

}

cout<<endl;

Search(g,lowcost,vex,k);

}

void Prim(Graph g,int k)//最小生成树prim算法

{

int m,n;

int visit[g.Ver+1];//访问标志

int lowcost[g.Ver+1];//权值

int vex[g.Ver+1];//存顶点

for(int i=1;i<=g.Ver;i++)

{

visit[i]=0;

lowcost[i]=g.arc[k][i];

vex[i]=k;

}

visit[k]=1;

int key=g.Ver-1;//循环次数

cout<<"平均能耗最小网络结构加入过程："<<endl;

while(key!=0)

{

int min;

int word;

for(int i=1;i<=g.Ver;i++)//设一个最小值

{

if(!visit[i])//未被访问的

{

min=lowcost[i];

word=i;

break;

}

}

for(int i=1;i<=g.Ver;i++)//找到权值最小值

{

if(!visit[i])

{

if(lowcost[i]<min)

{

min=lowcost[i];

word=i;//记录位置

}

}

}

visit[word]=1;//标记

key--;//次数减一

if(g.arc[vex[word]][word]!=Infine)

cout<<"<"<<vex[word]<<","<<word<<">"<<" ";//输出下一个节点

for(int j=1;j<=g.Ver;j++)//刷新权值数组

{

if(!visit[j])

{

if(g.arc[word][j]<lowcost[j])

{

lowcost[j]=g.arc[word][j];//权值改变

vex[j]=word;//顶点数组改变

}

}

}

}

}

int main()

{

cout<<" 无线传感器网络"<<endl;

cout<<" 1.创建文件"<<endl;

cout<<" 2.最短时间通道"<<endl;

cout<<" 3.平均能耗最小网络结构"<<endl;

int choice;

cout<<"please input your choice:";

cin>>choice;

switch(choice)

{

case 1:

system("cls");

cout<<" 无线传感器网络"<<endl;

cout<<" 1.创建文件"<<endl;

Creat\_map();

break;

case 2:

{

system("cls");

cout<<" 无线传感器网络"<<endl;

cout<<" 2.最短时间通道"<<endl;

Graph g;

Init\_Graph(g);

Creat\_Graph(g);

cout<<endl;

cout<<"初始图："<<endl;

Print\_Graph(g);

int data;

cout<<"please input the first number:";

cin>>data;

cout<<endl;

Short\_Path(g,data);

break;

}

case 3:

{

system("cls");

cout<<" 无线传感器网络"<<endl;

cout<<" 3.平均能耗最小网络结构"<<endl;

Graph g;

Init\_Graph(g);

Creat1\_Graph(g);

cout<<endl;

cout<<"初始图："<<endl;

Print\_Graph(g);

int data;

cout<<"please input the first number:";

cin>>data;

cout<<endl;

Prim(g,data);

break;

}

default:

cout<<"chioce input is error!"<<endl;

break;

}

return 0;

}

六、排序算法比较

1、题目简介

[问题描述]

利用随机函数产生10个样本的20000个随机整数（其中之一已经是正序，之一是逆序），利用直接插入排序、希尔排序，冒泡排序、快速排序、选择排序、堆排序，归并排序、基数排序8种排序方法进行排序（结果为由小到大的顺序），并统计每一种排序算法对不同样本所耗费的时间。

[基本要求]

（1） 原始数据存在文件中，用相同样本对不同算法进行测试；

（2） 屏幕显示每种排序算法对不同样本所花的时间；

2、数据结构

#define max 20001

#define min 10

typedef struct

{

int \*elem;

int length;

int listsize;

}SqList;

#define key 5

struct QNode

{

int data;

QNode \*next;

};

typedef struct LinkQueue

{

QNode \*front;

QNode \*rear;

}\*Link\_Queue;

1. 算法设计思想

（1）插入排序

思想：将序列分为【有序段】与【无序段】两段，然后依次将【无序段】中的元素插入到【有序段】的正确位置。寻找元素应插入的位置可用【直接】和【折半】两种方式进行查找。

（2）希尔排序（shell sort）

思想：提供一个增量序列（递减），每次根据增量序列将待排序的记录划分为若干组，然后分别对这若干组采用插入排序进行排序。当经过几次分组排序之后，记录的排列已经基本有序，这个时候再对所有的记录直接插入排序。

（3） 冒泡排序

思想：将序列分为【有序段】与【无序段】两段，将【无序段】的相邻元素进行比较，若逆序，则交换，最后将最右边的元素放入有序段。最终达到有序化。

（4）快速排序

思想：在待排序序列中选取一个记录作为基准值，然后从待排序序列左右两端开始，逐渐向中间靠拢，交替与基准值进行比较，交换。每次交换，当右侧记录小基准值，则将其与基准值交换，使其移到基准值的左侧；反之，当左侧记录大于基准值，则将其与基准值交换，使其移到基准值的右侧。最终根据基准值将序列分为左右两个子序列，然后对这两个子序列重复上面步骤。

（5） 简单选择排序

思想：从待排序的序列中选取出最小（最大）的元素作为有序序列新的元素。

（6）堆排序

思想：将待排序的记录序列构造一个堆，然后选出并移走该堆得最值，然后将剩余的记录再调节成堆，以此类推，知道堆中只有一个记录为止，每个记录出堆得顺序就是一个有序序列。

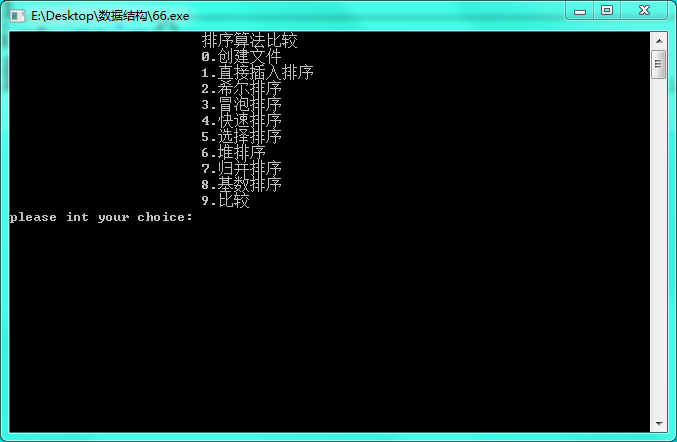
（7）归并排序

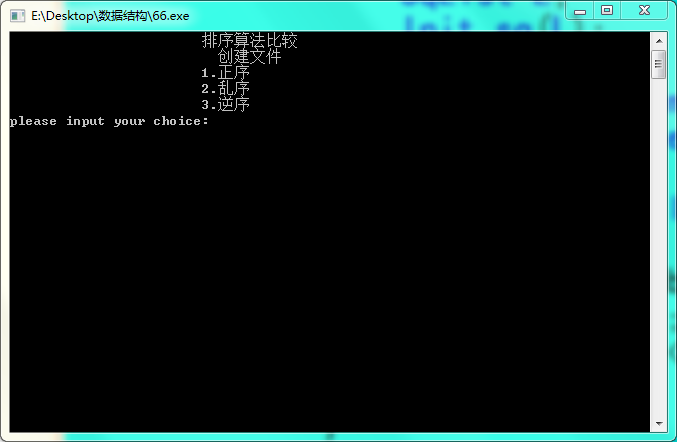
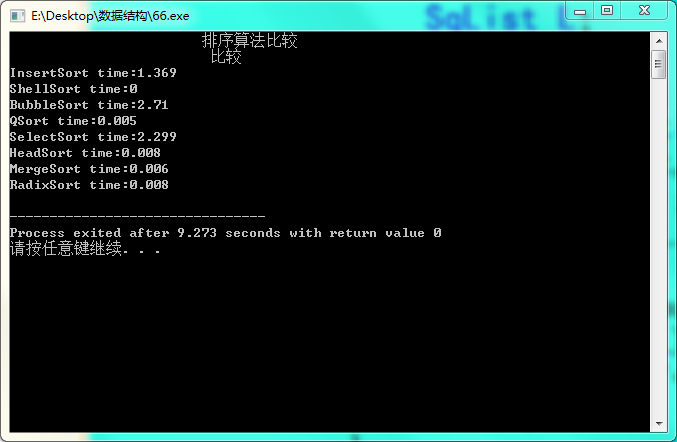
思想：将一个具有n个待排序记录的序列堪称是n个长度为1的有序列，然后进行两两归并，得到n/2个长度为2的有序序列，在进行两两归并，得到n/4个长度为4的有序序列，如此重复，直到得到一个长度为n的有序序列为止。

（8）基数排序

思想：按照关键字位进行“分配”和“收集”。

1. 测试数据和结果



选择文件：比较：其余的选择，我编写的是输出排序后的数据，并且求每一个时间，由于有2万个数据。这里省略。

5、算法时间复杂度

|  |  |  |
| --- | --- | --- |
| 排序法 | 时间复杂度 | 稳定度 |
| 冒泡 | O(n2) | 稳定 |
| 交换 | O(n2) | 不稳定 |
| 选择 | O(n2) | 不稳定 |
| 插入 | O(n2) | 稳定 |
| 基数 | O(logRB) | 稳定 |
| Shell | O(nlogn) | 不稳定 |
| 快速 | O(nlogn) | 不稳定 |
| 归并 | O(nlogn) | 稳定 |
| 堆 | O(nlogn) | 不稳定 |

6、源代码

#include<iostream>

using namespace std;

#include<malloc.h>

#include<ctime>

#include<cstdlib>

#include<fstream>

#include<string>

#define max 20001

#define min 10

typedef struct

{

int \*elem;

int length;

int listsize;

}SqList;

struct QNode

{

int data;

QNode \*next;

};

typedef struct LinkQueue

{

QNode \*front;

QNode \*rear;

}\*Link\_Queue;

bool InitQueue(Link\_Queue &Q);

void DestroyQueue(Link\_Queue Q);

void RadixSort(int \*L,int l\_len);//基数排序

void Distribute(QNode \*Node\_head,LinkQueue \*Queue,int i);//分配

void Collection(QNode \*Node\_head,LinkQueue \*Queue); //收集

int Get\_key(QNode \*p,int i);//得到位值

void Traverse(int \*L,int l\_len);

void Init\_sq(SqList &L);

void Creat\_sq(SqList &L,int n);//对顺序表初始化

void Traverse\_sq(SqList L);//遍历所有元素

void InsertSort(SqList &L);//简单插入排序

void ShellInsert(SqList &L,int dk);//希尔排序

void ShellSort(SqList &L,int dlta[],int t);//希尔排序

void BubbleSort(SqList &L);//冒泡排序

int Partion(SqList &L,int low,int high);//确定L.elem[i]元素的位置

void QSort(SqList &L,int low,int high);//快速排序

void SelectSort(SqList &L);//简单选择排序

void HeapAdjust(SqList &L,int s,int m);//大顶堆

void HeadSort(SqList &L);//堆排序

void Merge(SqList S,SqList &T,int s,int m,int t);//归并

void MergePass(SqList &S,SqList T,int n,int h);//一次归并并排序

void MergeSort(SqList &S,SqList T,int n); //归并排序

void fuzhi\_wen(SqList &L);//将文件中的文件内容传给顺序表

void creat\_wen();//正序文件

void creat1\_wen();//逆序文件

void creat1\_wen();//逆序文件

void Init\_sq(SqList &L)

{

L.elem=(int \*)malloc(max\*sizeof(int));

if(!L.elem)exit(0);

L.length=0;

L.listsize=max;

}

void Creat\_sq(SqList &L,int n)//对顺序表初始化

{

if(n>max)exit(0);

for(int i=1;i<=n;i++)

{

cin>>L.elem[i];

L.length++;

}

}

void Traverse\_sq(SqList L)//遍历所有元素

{

if (!L.elem)

return;

int i=1;

while(L.elem[i]&&i<=L.length)

{

if((i)%100==0)

{

cout<<endl;

}

cout<<L.elem[i]<<" ";

i++;

}

cout<<endl;

}

void InsertSort(SqList &L)//简单插入排序

{//在有序的序列里插入未有序的

for(int i=1;i<=L.length;i++)

{

if(L.elem[i]>L.elem[i-1])//如果大于有序的最后一个数，不用排序

continue;

else

{

int t=i-1;

while(L.elem[i]<L.elem[t])//循环找到待插位置

{

t--;

}

t++;//待插位置

int temp=L.elem[i];//记录临时

for(int j=i;j>t;j--)//后移

{

L.elem[j]=L.elem[j-1];

}

L.elem[t]=temp;

}

}

}

void ShellInsert(SqList &L,int dk)//希尔排序

{

for(int i=dk;i<=L.length;i++)//从dk+1开始

{

if(L.elem[i]<L.elem[i-dk])

{

int t=L.elem[i];//临时位置

int j=i-dk;

while(j>=0&&(t<L.elem[j]))//对所有间隔dk的数排序

{

L.elem[j+dk]=L.elem[j];

//记录后移，因为如果放在这个位置，会影响这个位置的希尔排序

j-=dk;

}

L.elem[j+dk]=t;

}

}

}

void ShellSort(SqList &L,int dlta[],int t)//希尔排序

{

for(int k=0;k<t;k++)

{

ShellInsert(L,dlta[k]);

}

}

void BubbleSort(SqList &L)//冒泡排序

{

for(int i=0;i<=L.length;i++)

for(int j=i+1;j<=L.length;j++)

{

if(L.elem[i]>L.elem[j])

{

int t=L.elem[i];

L.elem[i]=L.elem[j];

L.elem[j]=t;

}

}

}

int Partion(SqList &L,int low,int high)//确定L.elem[i]元素的位置

{

int i=low;

int j=high;

int t=L.elem[i];

while(i<j)

{

while(L.elem[j]>=t&&i<j)

{

j--;

}

L.elem[i]=L.elem[j];

while(L.elem[i]<=t&&i<j)

{

i++;

}

L.elem[j]=L.elem[i];

}

L.elem[i]=t;

return i;

}

void QSort(SqList &L,int low,int high)//快速排序

{

if(low<high)

{

int key=Partion(L,low,high);

QSort(L,low,key-1);

QSort(L,key+1,high);

}

}

void SelectSort(SqList &L)//简单选择排序

//在i之后选择一个最小的数置于i位置

{

for(int i=0;i<=L.length-1;i++)

{

int t=L.elem[i];

for(int j=i+1;j<=L.length;j++)

{

if(L.elem[j]<t)

{

int t1=L.elem[j];

L.elem[j]=t;

t=t1;

}

}

L.elem[i]=t;

}

}

void HeapAdjust(SqList &L,int s,int m)//大顶堆

{

int t=L.elem[s];

for(int j=2\*s;j<=m;j\*=2)//从非叶子节点向下搜索最小的值置于根

{

if(j<m&&(L.elem[j]<L.elem[j+1]))

j++;//比较左孩子右孩子，j指向较大者

if(t>L.elem[j])

break;//如果t比左右孩子都大

L.elem[s]=L.elem[j];//将s位置放置左右孩子的最大的值

s=j;//s下移到j 位置

}

L.elem[s]=t;//这里的s位置已变

}

void HeadSort(SqList &L)//堆排序

{

for(int i=L.length/2;i>0;i--)

HeapAdjust(L,i,L.length);

for(int j=L.length;j>1;j--)

{

int t=L.elem[1];

L.elem[1]=L.elem[j];

L.elem[j]=t;//将最大的值放在最后一个位置

HeapAdjust(L,1,j-1);//去除最后一个值重新创建大顶堆

}

}

void Merge(SqList S,SqList &T,int s,int m,int t)//归并

{//将S中的数据归并并排序T中 s为第一个有序序列起点 m+1为第二个有序序列起点

int i=s,j=m+1,k=s;//记录位置

while(i<=m&&j<=t)

{

if(S.elem[i]<=S.elem[j])

T.elem[k++]=S.elem[i++];

else

T.elem[k++]=S.elem[j++];

}

if(i<=m)

while(i<=m)

T.elem[k++]=S.elem[i++];

else while(j<=t)

T.elem[k++]=S.elem[j++];

}

void MergePass(SqList &S,SqList T,int n,int h)//一次归并并排序

{

int i=1;

while(i<=n-2\*h+1)//将序列分成h为单位的序列

{

Merge(S,T,i,i+h-1,i+2\*h-1);//归并0-h h-2h到

i+=2\*h;//每2h归并一次

}

if(i<n-h+1)Merge(S,T,i,i+h-1,n);//最后一个序列不是h 为单位的序列且小于h

else//剩下一个子序列没有归并

{

for(int k=i;k<=n;k++)

{

T.elem[k]=S.elem[k];

}

}

}

void MergeSort(SqList &S,SqList T,int n)

{

int h=1;//初始子序列长度为1 n为序列长度

while(h<n)

{

MergePass(S,T,n,h);//h的长度不断变化

h=2\*h;

MergePass(T,S,n,h);

h=2\*h;

}

}

#define key 5

bool InitQueue(LinkQueue \*Q)

{

Q->front=new QNode;

if(!(Q->front))

return false;

Q->front->next=NULL;

Q->front->data=-1;

Q->rear=Q->front;

return true;

}

void DestroyQueue(LinkQueue \*Q)

{

Q->rear=NULL;

delete Q->front;

Q->front=NULL;

}

void RadixSort(SqList &L,int l\_len)//基数排序

{

LinkQueue \*Queue=new LinkQueue[10];

QNode \*Node\_head=new QNode,\*p,\*pre\_p;

int i;

for(i=0;i<10;i++)

InitQueue(Queue+i);//构造桶

pre\_p=Node\_head;

for(i=1;i<=l\_len;i++)//顺序表转链表

{

p=new QNode;

p->data=L.elem[i];

pre\_p->next=p;

pre\_p=p;

}

pre\_p->next=NULL;

for(i=1;i<=key;i++)//分配、收集

{

Distribute(Node\_head,Queue,i);

Collection(Node\_head,Queue);

}

p=Node\_head->next;

for(i=1;i<=l\_len;i++,p=p->next)//链表信息写入顺序表

L.elem[i]=p->data;

pre\_p=Node\_head;

while(pre\_p)

{

p=pre\_p->next;

delete pre\_p;

pre\_p=p;

}

for(i=0;i<10;i++)

DestroyQueue(Queue+i);

delete []Queue;

}

void Distribute(QNode \*Node\_head,Link\_Queue Queue,int i)//分配

{

QNode \*p=Node\_head->next;

int t;

while(p)

{

t=Get\_key(p,i);

Queue[t].rear->next=p;

Queue[t].rear=p;

p=p->next;

Queue[t].rear->next=NULL;

}

Node\_head->next=NULL;

}

int Get\_key(QNode \*p,int i)//得到位数

{

int j,s=p->data;

for(j=1;j<i;j++)

s/=10;

return s%10;

}

void Collection(QNode \*Node\_head,Link\_Queue Queue)//收集

{

int i;

QNode \*p=Node\_head;

for(i=0;i<10;i++)

if(Queue[i].front->next)

{

p->next=Queue[i].front->next;

p=Queue[i].rear;

Queue[i].front->next=NULL;

Queue[i].rear=Queue[i].front;

}

}

void fuzhi\_wen(SqList &L)//将文件中的文件内容传给顺序表

{

fstream file;

file.open("paixu.txt",ios::in);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

int t,i=1;

file>>t;

L.length=0;

while(!file.eof())

{

L.elem[i]=t;

L.length++;

i++;

file>>t;

}

file.close();

}

void creat\_wen()//正序文件

{

fstream file;

file.open("paixu.txt",ios::out);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

for(int i=1;i<max;i++)

{

file<<i<<" ";

//cout<<i<<" ";

}

file.close();

// fstream file;

/\* file.open("paixu.txt",ios::in);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

int t;

file>>t;

while(!file.eof())

{

cout<<t<<" ";

file>>t;

}

file.close();\*\*/

}

void creat1\_wen()//逆序文件

{

fstream file;

file.open("paixu.txt",ios::out);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

int t=max-1;

for(int i=1;i<max;i++)

{

file<<t<<" ";

t--;

}

file.close();

}

void creat2\_wen()//乱序文件

{

fstream file;

file.open("paixu.txt",ios::out);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

int t;

srand((unsigned)time(NULL));

t=rand()%20000;

for(int i=1;i<max;i++)

{

file<<t<<" ";

t=rand()%20000;

}

file.close();

}

int main()

{//直接插入排序、希尔排序，冒泡排序、快速排序、选择排序、堆排序，归并排序、基数排序

cout<<" 排序算法比较"<<endl;

cout<<" 0.创建文件"<<endl;

cout<<" 1.直接插入排序"<<endl;

cout<<" 2.希尔排序"<<endl;

cout<<" 3.冒泡排序"<<endl;

cout<<" 4.快速排序"<<endl;

cout<<" 5.选择排序"<<endl;

cout<<" 6.堆排序"<<endl;

cout<<" 7.归并排序"<<endl;

cout<<" 8.基数排序"<<endl;

cout<<" 9.比较"<<endl;

int chioce;

cout<<"please int your choice:";

cin>>chioce;

switch(chioce)

{

case 0:

{

system("cls");

cout<<" 排序算法比较"<<endl;

cout<<" 创建文件"<<endl;

cout<<" 1.正序"<<endl;

cout<<" 2.乱序"<<endl;

cout<<" 3.逆序"<<endl;

int c;

cout<<"please input your choice:";

cin>>c;

switch(c)

{

case 1:

creat\_wen();

break;

case 2:

creat2\_wen();

break;

case 3:

creat1\_wen();

break;

default:

cout<<"chioce input is error!"<<endl;

break;

}

/\* SqList L;

Init\_sq(L);

fuzhi\_wen(L);

Traverse\_sq(L);\*/

break;

}

case 1:

{

system("cls");

cout<<" 排序算法比较"<<endl;

cout<<" 直接插入排序"<<endl;

SqList L;

Init\_sq(L);

fuzhi\_wen(L);

double cost\_time;

clock\_t start,end;

start=clock();

InsertSort(L);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

Traverse\_sq(L);

cout<<endl;

cout<<"time:"<<cost\_time<<endl;

break;

}

case 2:

{

system("cls");

cout<<" 排序算法比较"<<endl;

cout<<" 希尔排序"<<endl;

SqList L;

Init\_sq(L);

int a[3]={5,3,1};

fuzhi\_wen(L);

double cost\_time;

clock\_t start,end;

start=clock();

ShellSort(L,a,3);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

Traverse\_sq(L);

cout<<"time:"<<cost\_time<<endl;

break;

}

case 3:

{

system("cls");

cout<<" 排序算法比较"<<endl;

cout<<" 冒泡排序"<<endl;

SqList L;

Init\_sq(L);

fuzhi\_wen(L);

double cost\_time;

clock\_t start,end;

start=clock();

BubbleSort(L);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

Traverse\_sq(L);

cout<<"time:"<<cost\_time<<endl;

break;

}

case 4:

{

system("cls");

cout<<" 排序算法比较"<<endl;

cout<<" 快速排序"<<endl;

SqList L;

Init\_sq(L);

fuzhi\_wen(L);

double cost\_time;

clock\_t start,end;

start=clock();

QSort(L,0,L.length);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

Traverse\_sq(L);

cout<<"time:"<<cost\_time<<endl;

break;

}

case 5:

{

system("cls");

cout<<" 排序算法比较"<<endl;

cout<<" 选择排序"<<endl;

SqList L;

Init\_sq(L);

fuzhi\_wen(L);

double cost\_time;

clock\_t start,end;

start=clock();

SelectSort(L);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

Traverse\_sq(L);

cout<<"time:"<<cost\_time<<endl;

break;

}

case 6:

{

system("cls");

cout<<" 排序算法比较"<<endl;

cout<<" 堆排序"<<endl;

SqList L;

Init\_sq(L);

fuzhi\_wen(L);

double cost\_time;

clock\_t start,end;

start=clock();

HeadSort(L);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

Traverse\_sq(L);

cout<<"time:"<<cost\_time<<endl;

break;

}

case 7:

{

system("cls");

cout<<" 排序算法比较"<<endl;

cout<<" 归并排序"<<endl;

SqList L,L1;

Init\_sq(L);

Init\_sq(L1);

fuzhi\_wen(L);

for(int i=1;i<=L.length;i++)

{

L1.elem[i]=L.elem[i];

}

L1.length=L.length;

double cost\_time;

clock\_t start,end;

start=clock();

MergeSort(L,L1,L.length);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

Traverse\_sq(L);

cout<<"time:"<<cost\_time<<endl;

break;

}

case 8:

{

system("cls");

cout<<" 排序算法比较"<<endl;

cout<<" 基数排序"<<endl;

SqList L;

Init\_sq(L);

fuzhi\_wen(L);

double cost\_time;

clock\_t start,end;

start=clock();

RadixSort(L,L.length);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

Traverse\_sq(L);

cout<<"time:"<<cost\_time<<endl;

break;

}

case 9:

{

system("cls");

cout<<" 排序算法比较"<<endl;

cout<<" 比较"<<endl;

SqList L,L1;

Init\_sq(L);

fuzhi\_wen(L);

double cost\_time;

clock\_t start,end;

start=clock();

InsertSort(L);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

cout<<"InsertSort time:"<<cost\_time<<endl;

int a[3]={5,3,1};

fuzhi\_wen(L);

start=clock();

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

cout<<"ShellSort time:"<<cost\_time<<endl;

fuzhi\_wen(L);

start=clock();

BubbleSort(L);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

cout<<"BubbleSort time:"<<cost\_time<<endl;

fuzhi\_wen(L);

start=clock();

QSort(L,0,L.length);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

cout<<"QSort time:"<<cost\_time<<endl;

fuzhi\_wen(L);

start=clock();

SelectSort(L);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

cout<<"SelectSort time:"<<cost\_time<<endl;

fuzhi\_wen(L);

start=clock();

HeadSort(L);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

cout<<"HeadSort time:"<<cost\_time<<endl;

Init\_sq(L1);

fuzhi\_wen(L);

for(int i=1;i<=L.length;i++)

{

L1.elem[i]=L.elem[i];

}

L1.length=L.length;

start=clock();

MergeSort(L,L1,L.length);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

cout<<"MergeSort time:"<<cost\_time<<endl;

fuzhi\_wen(L);

start=clock();

RadixSort(L,L.length);

end=clock();

cost\_time=(double) (end-start)/(CLOCKS\_PER\_SEC);

cout<<"RadixSort time:"<<cost\_time<<endl;

break;

}

default:

cout<<"chioce input is error!"<<endl;

break;

}

return 0;

}

七、Hash表应用

1、题目简介

[问题描述]

设计散列表实现VIP客户发掘。对身份证号进行Hash, 通过对乘客某时间段内的乘机频率、里程数统计，发掘VIP客户。

[基本要求]

（1） 设每个记录有下列数据项：身份证号码（虚构，位数和编码规则与真实一致即可）、姓名、航班号、航班日期、里程。

（2） 从文件输入各记录，以身份证号码为关键字建立散列表。

（3） 分别采用开放定址（自行选择和设计定址方案）和链地址两种方案解决冲突；显示发生冲突的次数、每次中解决冲突进行重定位的次数。

（4）记录条数至少在100条以上。

（5） 从记录中实现乘客乘机频率、里程数统计，从而发掘VIP客户。

2、数据结构

开放定址法：

typedef struct hash//hash表结构体

{

int \*elem;

int length;

int listsize;

}hash;

typedef struct people//记录结构体

{

int ID;//身份证号码

char name[10];//姓名

int code;//航班号

int data;//航班日期

int distance;//里程

int vip;//VIP

}people;

链地址法：

typedef struct Hash//hash表结构体

{

int elem;

struct Hash \*next;

}Hash,\*LHash;

typedef struct H

{

int data;

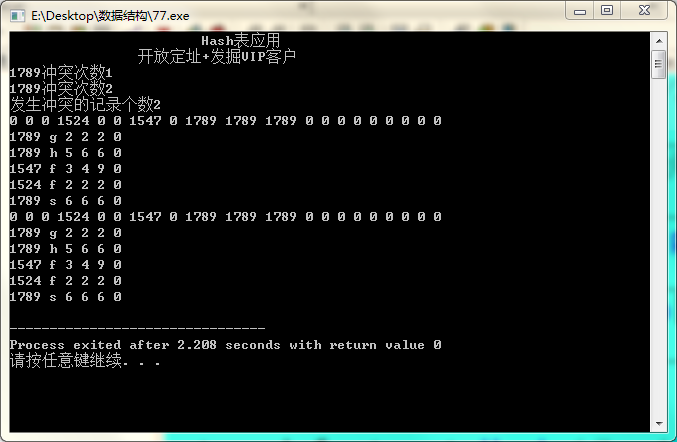
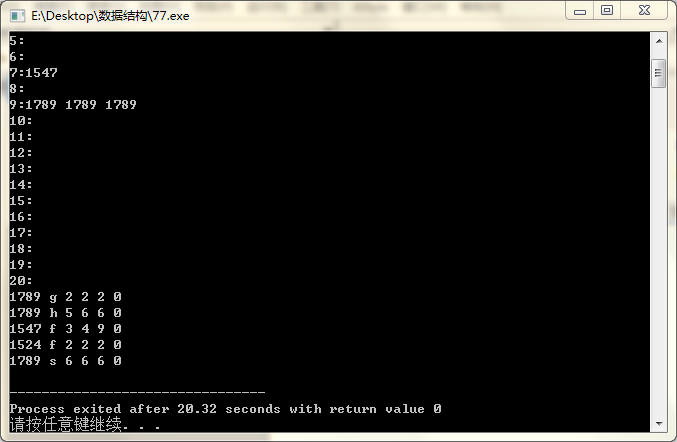
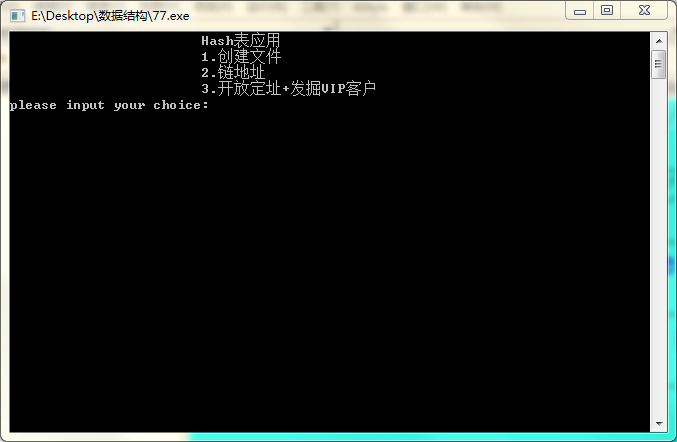
Hash \*first;

}H;

1. 算法设计思想

开放定址法：当冲突发生时，使用某种探查(亦称探测)技术在散列表中形成一个探查(测)序列。沿此序列逐个单元地查找，直到找到给定的关键字，或者碰到一个开放的地址(即该地址单元为空)为止（若要插入，在探查到开放的地址，则可将待插入的新结点存人该地址单元）。查找时探查到开放的 地址则表明表中无待查的关键字，即查找失败。

链地址法：将所有关键字为同义词的结点链接在同一个单链表中。若选定的散列表长度为m，则可将散列表定义为一个由m个头指针组成的指针数 组T[0..m-1]。凡是散列地址为i的结点，均插入到以T[i]为头指针的单链表中。T中各分量的初值均应为空指针。

4、测试数据和结果

5、算法时间复杂度

时间复杂度0（n）

改进：链地址法找VIP需要实现

6、源代码

#include<iostream>

using namespace std;

#include<malloc.h>

#include<fstream>

#include<string.h>

#define maxsize 20

#define minsize 1

#define max 1000

typedef struct hash//hash表结构体

{

int \*elem;

int length;

int listsize;

}hash;

typedef struct people//记录结构体

{

int ID;//身份证号码

char name[10];//姓名

int code;//航班号

int data;//航班日期

int distance;//里程

int vip;//VIP

}people;

typedef struct Hash//hash表结构体

{

int elem;

struct Hash \*next;

}Hash,\*LHash;

typedef struct H

{

int data;

Hash \*first;

}H;

void Init\_Hash(hash &L);

void Search\_VIP(people p[],hash L,int n);//找vip客户

void creat\_wen();//创建记录文件

void Print\_people(people p[],int n);

void Creat\_kHash(people p[]);//开放定址

void Print\_LHash(H L[]);//打印哈希表

void Print\_Hash(hash L);//打印哈希表

void Creat\_LHash(people p[]);//链地址

void Init\_Hash(hash &L)

{

L.elem=(int \*)malloc(maxsize\*sizeof(int));

if(!L.elem)exit(0);

L.length=0;

L.listsize=maxsize;

}

void Search\_VIP(people p[],hash L,int n)//找vip客户

{

for(int i=1;i<=L.listsize;i++ )

{

int count,mdistance;

if(L.elem[i]!=0)//hash表中不空

{

count=1;//频率

int k;

for(int s=1;s<=n;s++ )//找到对应的客户，记录位置

{

if(p[s].ID==L.elem[i])

{

k=s;

break;

}

}

mdistance=p[k].distance;//里程数统计

for(int j=i+1;j<=n;j++)

{

if(L.elem[j]==L.elem[i])

{

count++;

L.elem[j]=0;

for(int s=1;s<=n;s++ )

{

if(p[s].ID==L.elem[j])

{

k=s;

break;

}

}

mdistance+=p[k].distance;

}

}

}

for(int s=1;s<=n;s++ )

{

if(p[s].ID==L.elem[i])

{

if(count>=4&&count<8)//根据频率发掘VIP

p[s].vip=1;

else if(count>=8)

{

p[s].vip=2;

}

}

if(mdistance>100)//根据里程量发掘VIP

p[s].vip=1;

else if(mdistance>200)

p[s].vip=2;

}

}

}

void Print\_Hash(hash L)//打印哈希表

{

for(int i=1;i<=L.listsize;i++ )

{

cout<<L.elem[i]<<" ";

}

cout<<endl;

}

void creat\_wen()//创建记录文件

{

fstream file;

file.open("hash.dat",ios::out|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

int n;

cout<<"please input n:";//创建记录

cin>>n;

people p[max];

cout<<"please input every information:"<<endl;//输入记录

cout<<"ID name code data distance:"<<endl;

for(int i=1;i<=n;i++ )

{

cin>>p[i].ID;

cin>>p[i].name;

cin>>p[i].code;

cin>>p[i].data;

cin>>p[i].distance;

p[i].vip=0;

file.write((char \*)&p[i],sizeof(p[i]));

}

file.close();

people f;

file.open("hash.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

while(!file.eof())

{

cout<<f.ID<<" ";

cout<<f.name<<" ";

cout<<f.code<<" ";

cout<<f.data<<" ";

cout<<f.distance<<" ";

cout<<f.vip<<endl;

file.read((char \*)&f,sizeof(f));

}

file.close();

}

void Print\_people(people p[],int n)

{

for(int i=1;i<=n;i++ )

{

cout<<p[i].ID<<" ";

cout<<p[i].name<<" ";

cout<<p[i].code<<" ";

cout<<p[i].data<<" ";

cout<<p[i].distance<<" ";

cout<<p[i].vip<<endl;

}

}

void Creat\_kHash(people p[])//开放定址

{

fstream file;

people f;

file.open("hash.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

int n=0,i=1;

while(!file.eof())

{

p[i].ID=f.ID;

strcpy(p[i].name,f.name);

p[i].code=f.code;

p[i].data=f.data;

p[i].distance=f.distance;

p[i].vip=f.vip;

i++;

n++;

file.read((char \*)&f,sizeof(f));

}

file.close();

/\* int n;

cout<<"please input n:";//创建记录

cin>>n;

cout<<"please input every information:"<<endl;//输入记录

cout<<"ID name code data distance:"<<endl;

for(int i=1;i<=n;i++ )

{

cin>>p[i].ID;

cin>>p[i].name;

cin>>p[i].code;

cin>>p[i].data;

cin>>p[i].distance;

p[i].vip=0;

}\*/

hash L;//建立hash表和初始化

Init\_Hash(L);

for(int i=1;i<=L.listsize;i++ )

{

L.elem[i]=0;

}

int count=0;//count记录发生冲突的记录个数

for(int i=1;i<=n;i++)

{

int t=p[i].ID%10;

if(L.elem[t]!=0)

{

count++;

t++;

int key=1;//key记录某个数据发生冲突的次数

while(L.elem[t]!=0)

{

key++;

t++;//采用线性探测再散列

}

L.elem[t]=p[i].ID;

cout<<p[i].ID<<"冲突次数"<<key<<endl;

}

else

{

L.elem[t]=p[i].ID;

}

}

cout<<"发生冲突的记录个数"<<count<<endl;

Print\_Hash(L);

Print\_people(p,n);

Search\_VIP(p,L,n);

Print\_Hash(L);

Print\_people(p,n);

}

void Print\_LHash(H L[])//打印哈希表

{

for(int i=1;i<=maxsize;i++ )

{

Hash \*p;

cout<<L[i].data<<":";

p=L[i].first;

while(p)

{

cout<<p->elem<<" ";

p=p->next;

}

cout<<endl;

}

}

void Creat\_LHash(people p[])//链地址

{

fstream file;

people f;

file.open("hash.dat",ios::in|ios::binary);

if(file.fail())

{

cout<<"file open is error!"<<endl;

exit(0);

}

file.read((char \*)&f,sizeof(f));

int n=0,i=1;

while(!file.eof())

{

p[i].ID=f.ID;

strcpy(p[i].name,f.name);

p[i].code=f.code;

p[i].data=f.data;

p[i].distance=f.distance;

p[i].vip=f.vip;

i++;

n++;

file.read((char \*)&f,sizeof(f));

}

file.close();

/\* int n;

cout<<"please input n:";//创建记录

cin>>n;

cout<<"please input every information:"<<endl;//输入记录

cout<<"ID name code data distance:"<<endl;

for(int i=1;i<=n;i++ )

{

cin>>p[i].ID;

cin>>p[i].name;

cin>>p[i].code;

cin>>p[i].data;

cin>>p[i].distance;

p[i].vip=0;

}\*/

H L[max];//建立hash表和初始化

for(int i=1;i<=maxsize;i++ )

{

L[i].data=i;

L[i].first=NULL;

}

int count=0;//count记录发生冲突的记录个数

for(int i=1;i<=n;i++)

{

int t=p[i].ID%10;

if(L[t].first!=NULL)

{

count++;

Hash \*q,\*s,\*p1=L[t].first;

s=(Hash \*)malloc(sizeof(Hash));

s->elem=p[i].ID;

s->next=NULL;

while(p1)

{

q=p1;

p1=p1->next;

}

q->next=s;

;

}

else

{

Hash \*q=(Hash \*)malloc(sizeof(Hash));

q->elem =p[i].ID;

L[t].first=q;

q->next=NULL;

}

}

cout<<"发生冲突的记录个数"<<count<<endl;

Print\_LHash(L);

Print\_people(p,n);

}

int main()

{

cout<<" Hash表应用"<<endl;

cout<<" 1.创建文件"<<endl;

cout<<" 2.链地址"<<endl;

cout<<" 3.开放定址+发掘VIP客户"<<endl;

int choice;

cout<<"please input your choice:";

cin>>choice;

switch(choice)

{

case 1:

system("cls");

cout<<" Hash表应用"<<endl;

cout<<" 创建文件"<<endl;

creat\_wen();

break;

case 2:

system("cls");

cout<<" Hash表应用"<<endl;

cout<<" 链地址"<<endl;

people p1[max];

Creat\_LHash(p1);

break;

case 3:

system("cls");

cout<<" Hash表应用"<<endl;

cout<<" 开放定址+发掘VIP客户"<<endl;

people p[max];

Creat\_kHash(p);

break;

default:

cout<<"chioce input is error!"<<endl;

break;

}

return 0;

}

八、总结

六个必做题，都全部实现，只是有一些瑕疵（高老师验收时指出的），现在基本已经改正。最不好的地方就是Huffman树的储存，最苦恼自己。

自己又做了第七个题，也成功实现。自己也尝试了算术表达式求值这个题，只实现了一部分功能。

代码量：4000左右

九、心得体会

见下一页。

心得体会

这次课设是一次痛苦但又快乐的“旅行”；

痛苦于花了时间并不一定做得出来。需要自己一步一步的检查与分析。因为每一题，你仔细读，你会发现言简意深。每个小问都隐藏着很大的知识点，隐藏着一些需要我们听说过但没有用的知识来实现，像如何截取函数运行时间，压缩存文件。对于一个对空间概论迷糊的我来说，就是‘煎熬’。有很多东西不清楚，或者根本不会这些东西。幸好我的舍友李荣毅，他的数据结构很强，也是我们的班小辅导师，帮忙我解决了很多问题。这里当然很感谢他了。

最多的程序代码当属第一题，因为开始对链表的组合即有两个指针，比较难理解。自己尝试的写了很久，甚至画分析结构图。到电脑上写代码时，发现到处出错。自己也查了一些资料，但网上的资料太杂了，并没有解决自己的问题。我并没有没想过问别人，我只是想自己解决。最后也没有成功。这就是花了时间也不一定做得出来，最后还是舍友帮我解决的，最后发现原来是空间的申请与释放出问题和一些其他的小错误。下面的每一题也差不多会出现这样的问题。

痛苦于抛开书本，就感觉自己什么都不会。很多的题不是我们不知道，而是我们无法实现，就像排序一样。我们一看书，都知道是哪个思想，可叫我们自己去实现时，整个人都是懵的。这里也必须承认，我自己的课设有好几部分是将书上的伪代码转换过来的。这里我收获的是重新复习了一遍知识点，失去的却是自己动手能力。弊大于利。高老师也多次强调这里，这里是我自己做不好。在高老师验收时，也向我问了这些问题，好像不止问了我。发现有很多人像我一样。这里，我也会重新审视自己，尽量尽力的自己动手，少依靠课本。多去发现问题、解决问题。将课本的知识变成自己的东西。

痛苦虽多，但快乐，或者收获更多。

快乐于一场知识的遨游，整体上对这个学期的课程内容复习与整合了一遍。并且脑补了一些课外知识。虽然编程是一件繁琐而又枯燥的事情，但如果你真正用心的去编，你会发现很多乐趣的。就像迷宫问题，你可以采取多种方法去实现，可以自己定义出口与入口，可以用很多方式实现地图界面，这当然取决于我们自己的能力，与兴趣程度。还有书上的八皇后问题，我就认为是一个很有趣的问题，我自己就自己尝试过，但那时给了一个定点。也成功了。现在，我可以实现很多种了。

快乐于对自己的一种重新认知与定义。这个得谈起验收。我被老师抽中了。虽然是伪随机的，可能应该是上次习题课作业，我交了但没写名字。就被幸运的抽中了。这个我和学长发消息说过了，可能高老师不知道吧。我并没有其他的意思，其实我还很感谢这次抽中。高老师与我的交谈中，让我认识到了很多问题，一是不要过度依赖他人，因为编程本来就容易出错，我就属于那种人。而每次发现错，自己只是简单看看哪里出错，并没有太多的认真检查。就去问别人。二是太多依赖课本了，并没有真正意义上的实现与理解它。三是要自己思考问题与解决问题。像排序，截取时间。我确实是问了舍友。自己并没有自己真正思考过，上网查过。高老师当场指出了这个问题。这里很感谢他。最后是读题、析题的仔细。不要自己写的程序与题目相差很大。我们应该准确的表达题目所给的。以上种种，确实是我自己的原因，除了一些客观原因，大部分是自己做不好。自己需要改正。

快乐于一种‘成就感’。每一个付出，不谈回报，但大大小小我们有一定的收获。我们每一个人也熬过夜写代码。也知其中味。就在验收得前一天晚上，我和舍友李荣毅就熬夜写了一个晚上，我主要是对代码的美化和拓展。他好像在写选做题。挺佩服他的。当早上的时候我写得差不多了，我那时就感觉有一种成就感，油然而生。在哪个时候，也知道熬夜多么的痛苦，睡觉是多么的美好。我和舍友就去吃了一个快乐的励志早餐。但感觉还是一种收获的快乐吧。之后，回来，我就整理了一下自己写的代码，就直接睡到要验收得哪个时候。而舍友则继续写代码。他花的时间也比较多，再加上大家请他去调错。我也相信他也会有这种收获的喜悦的。唯一有些遗憾的是自己没能够多做些，就像算术表达式求值只写了一半，并没有全部实现。这几天我应该可以把它写出来的。很多东西，我们不能太过于在意花在它上面的时间，因为它值得我们去花。

没有痛苦，哪里有快乐；没有付出，哪有有收获；只有我们动手做了，我们才知道最后成功的喜悦。对于很多东西，我们不仅要掌握其原理，而且还要把它实现。究其理，探其真。用习主席的一句话就是：“实干兴邦，甩开膀子，干”。我相信我自己也能够做到自己所希望的那个样子。最后，也再一次感谢高老师的指导和舍友在这次课设中的帮助。谢谢了。